算法设计与分析(2025年春季学期)
Divide-and-Conquer

授课老师: 栗师

南京大学计算机学院

Outline

1. Divide-and-Conquer
2. Counting Inversions
3. Solving Recurrences
4. Quicksort and Selection
    - Quicksort
    - Lower Bound for Comparison-Based Sorting Algorithms
    - Selection Problem
5. Polynomial Multiplication
6. Strassen's Algorithm for Matrix Multiplication
7. FFT(Fast Fourier Transform): Polynomial Multiplication in $O(n \log n)$ Time
8. Finding Closest Pair of Points in 2D Euclidean Space
9. Computing n-th Fibonacci Number

Greedy Algorithm

- mainly for combinatorial optimization problems
- trivial algorithm runs in exponential time
- greedy algorithm gives an efficient algorithm
- main focus of analysis: correctness of algorithm

Greedy Algorithm

- mainly for combinatorial optimization problems
- trivial algorithm runs in exponential time
- greedy algorithm gives an efficient algorithm
- main focus of analysis: correctness of algorithm

Divide-and-Conquer

- not necessarily for combinatorial optimization problems
- trivial algorithm already runs in polynomial time
- divide-and-conquer gives a more efficient algorithm
- main focus of analysis: running time

Divide-and-Conquer

- **Divide**: Divide instance into many smaller instances
- **Conquer**: Solve each of smaller instances recursively and separately
- **Combine**: Combine solutions to small instances to obtain a solution for the original big instance

merge-sort(A, n)

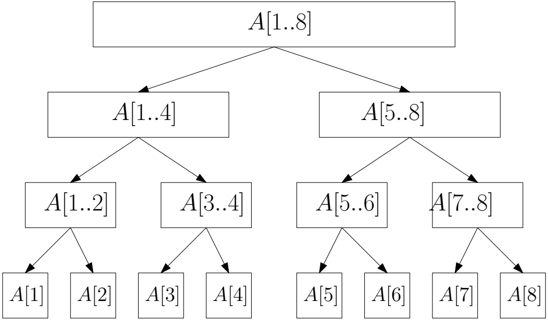
```
if $n = 1$ then
    return $A$
else
    $B \leftarrow$ merge-sort$\left(A\left[1..\lfloor n/2\rfloor\right], \lfloor n/2\rfloor\right)$
    $C \leftarrow$ merge-sort$\left(A\left[\lfloor n/2\rfloor + 1..n\right], \lceil n/2\rceil\right)$
    return merge$(B, C, \lfloor n/2\rfloor, \lceil n/2\rceil)$
```

merge-sort(A, n)

```
1: if $n = 1$ then
2:     return $A$
3: else
4:     $B \leftarrow$ merge-sort$\left(A\left[1..\lfloor n/2 \rfloor\right], \lfloor n/2 \rfloor\right)$
5:     $C \leftarrow$ merge-sort$\left(A\left[\lfloor n/2 \rfloor + 1..n\right], \lceil n/2 \rceil\right)$
6:     return merge$(B, C, \lfloor n/2 \rfloor, \lceil n/2 \rceil)$
```

- Divide: trivial
- Conquer: 4, 5
- Combine: 6

Running Time for Merge-Sort



- Each level takes running time $O(n)$
- There are $O(\log n)$ levels
- Running time $= O(n \log n)$
- Better than insertion sort

Running Time for Merge-Sort Using Recurrence

- $T(n) =$ running time for sorting n numbers,then

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) & \text{if } n \geq 2 \end{cases}$$

Running Time for Merge-Sort Using Recurrence

- $T(n) =$ running time for sorting n numbers,then

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) & \text{if } n \geq 2 \end{cases}$$

- With some tolerance of informality:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n \geq 2 \end{cases}$$

Running Time for Merge-Sort Using Recurrence

- $T(n) =$ running time for sorting n numbers,then

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) & \text{if } n \geq 2 \end{cases}$$

- With some tolerance of informality:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n \geq 2 \end{cases}$$

- Even simpler: $T(n) = 2T(n/2) + O(n)$. (Implicit assumption: $T(n) = O(1)$ if n is at most some constant.)

Running Time for Merge-Sort Using Recurrence

- $T(n) =$ running time for sorting n numbers,then

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) & \text{if } n \geq 2 \end{cases}$$

- With some tolerance of informality:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n \geq 2 \end{cases}$$

- Even simpler: $T(n) = 2T(n/2) + O(n)$. (Implicit assumption: $T(n) = O(1)$ if n is at most some constant.)
- Solving this recurrence, we have $T(n) = O(n \log n)$ (we shall show how later)

Outline

1. Divide-and-Conquer
2. Counting Inversions
3. Solving Recurrences
4. Quicksort and Selection
   - Quicksort
   - Lower Bound for Comparison-Based Sorting Algorithms
   - Selection Problem
5. Polynomial Multiplication
6. Strassen's Algorithm for Matrix Multiplication
7. FFT(Fast Fourier Transform): Polynomial Multiplication in $O(n \log n)$ Time
8. Finding Closest Pair of Points in 2D Euclidean Space
9. Computing n-th Fibonacci Number

Def. Given an array A of n integers, an inversion in A is a pair (i, j) of indices such that $i < j$ and $A[i] > A[j]$.

Def. Given an array A of n integers, an inversion in A is a pair (i, j) of indices such that $i < j$ and $A[i] > A[j]$.

Counting Inversions

Input: an sequence A of n numbers

Output: number of inversions in A

Def. Given an array A of n integers, an inversion in A is a pair (i, j) of indices such that $i < j$ and $A[i] > A[j]$.

Counting Inversions

Input: an sequence A of n numbers

Output: number of inversions in A

Example:

$$10 \qquad 8 \qquad 15 \qquad 9 \qquad 12$$

Def. Given an array A of n integers, an inversion in A is a pair (i, j) of indices such that $i < j$ and $A[i] > A[j]$.

Counting Inversions

Input: an sequence A of n numbers

Output: number of inversions in A

Example:

10	8	15	9	12
8	9	10	12	15

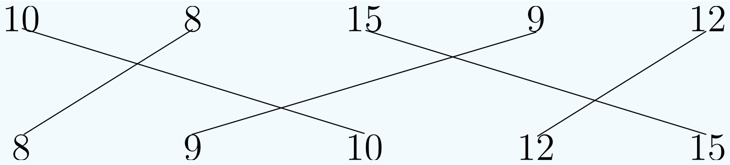
Def. Given an array A of n integers, an inversion in A is a pair (i, j) of indices such that $i < j$ and $A[i] > A[j]$.

Counting Inversions

Input: an sequence A of n numbers

Output: number of inversions in A

Example:



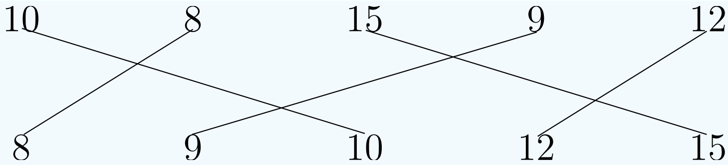
Def. Given an array A of n integers, an inversion in A is a pair (i, j) of indices such that $i < j$ and $A[i] > A[j]$.

Counting Inversions

Input: an sequence A of n numbers

Output: number of inversions in A

Example:



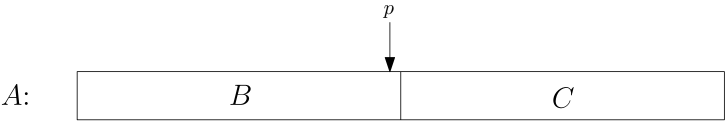
- 4 inversions (for convenience, using numbers, not indices): $(10, 8), (10, 9), (15, 9), (15, 12)$

Naive Algorithm for Counting Inversions

count-inversions(A, n)

```
$c \leftarrow 0$
for every $i \leftarrow 1$ to $n - 1$ do
    for every $j \leftarrow i + 1$ to $n$ do
        if $A[i] > A[j]$ then $c \leftarrow c + 1$
return $c$
```

Divide-and-Conquer



p

A: B C

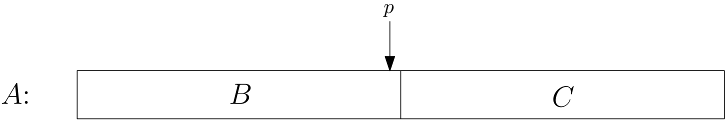
- $p = \lfloor n/2 \rfloor, B = A[1..p], C = A[p+1..n]$
- $$\#\text{invs}(A) = \#\text{invs}(B) + \#\text{invs}(C) + m$$ $$m = \left|\left\{(i,j) : B[i] > C[j]\right\}\right|$$

Q: How fast can we compute m, via trivial algorithm?

A: $O(n^2)$

- Can not improve the $O(n^2)$ time for counting inversions.

Divide-and-Conquer



- $p = \lfloor n/2 \rfloor, B = A[1..p], C = A[p+1..n]$
- $$\#\text{invs}(A) = \#\text{invs}(B) + \#\text{invs}(C) + m$$
  $$m = \big|\{(i,j) : B[i] > C[j]\}\big|$$

Lemma If both B and C are sorted, then we can compute m in $O(n)$ time!

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:

B:

3	8	12	20	32	48

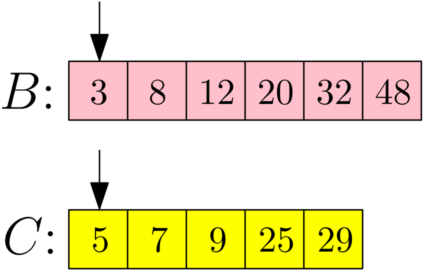
total$=$ 0

C:

5	7	9	25	29

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



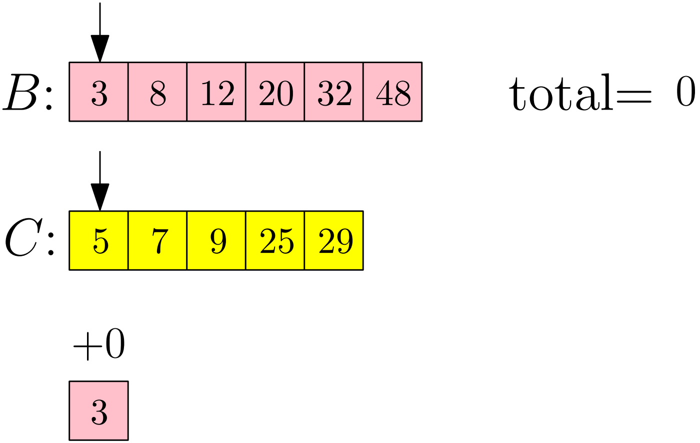
B: 3 8 12 20 32 48

C: 5 7 9 25 29

total$=$ 0

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B:	3	8	12	20	32	48

total= 0

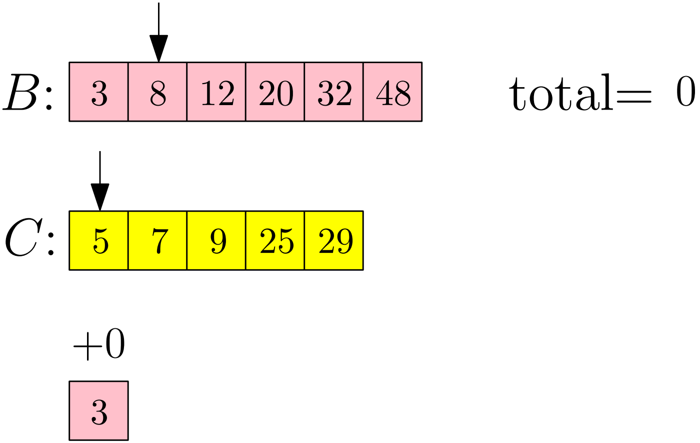
C:	5	7	9	25	29

$+0$

3

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B:	3	8	12	20	32	48
C:	5	7	9	25	29	

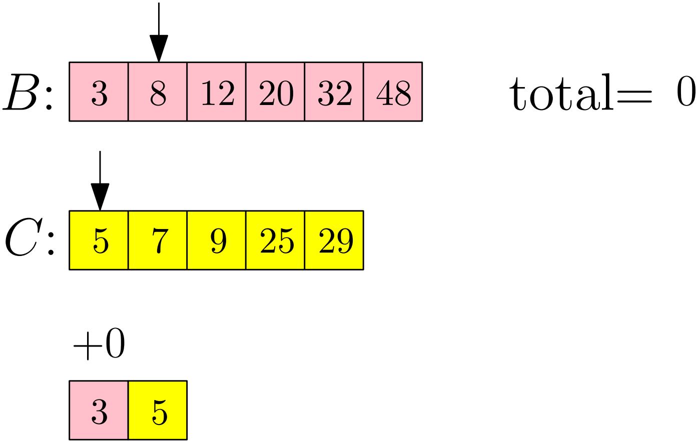
total$=$ 0

$+0$

3

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B: 3 8 12 20 32 48

total$=$ 0

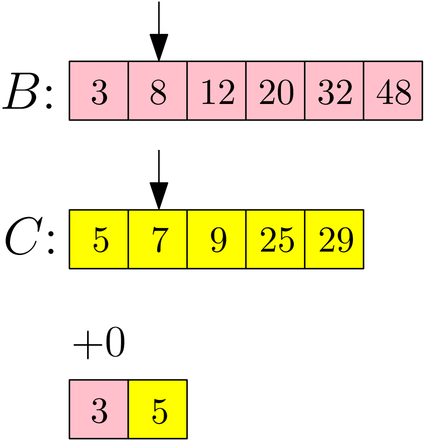
C: 5 7 9 25 29

$+0$

3 5

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B:

3	8	12	20	32	48

C:

5	7	9	25	29

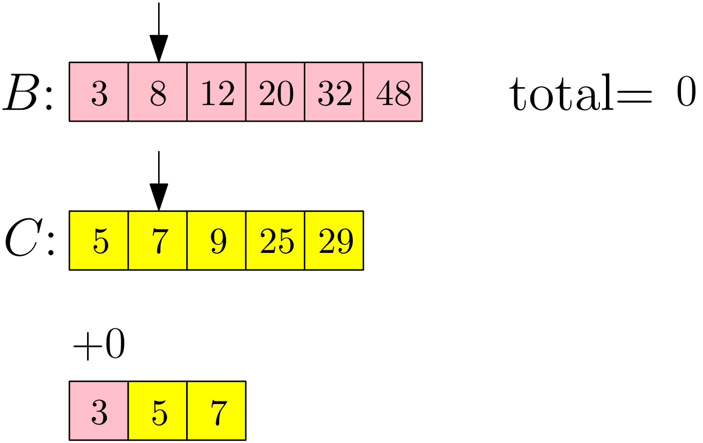
$+0$

3	5

total$=$ 0

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B: 3 8 12 20 32 48

total $= 0$

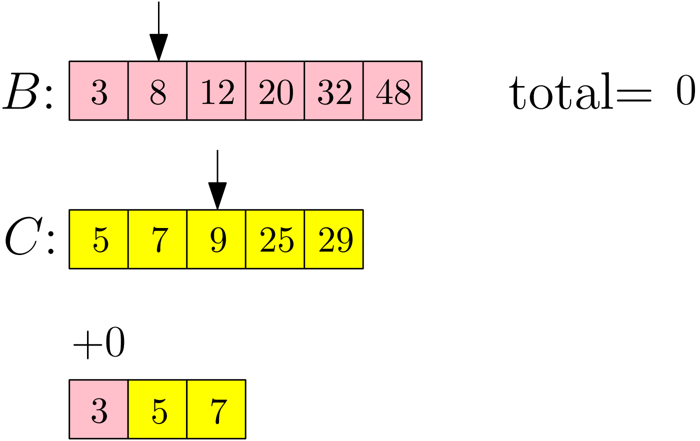
C: 5 7 9 25 29

$+0$

3 5 7

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B:

3	8	12	20	32	48

total$=$ 0

C:

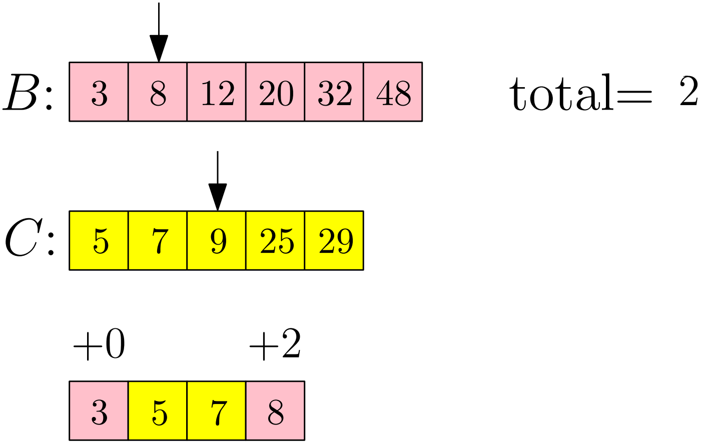
5	7	9	25	29

$+0$

3	5	7

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B: 3 | 8 | 12 | 20 | 32 | 48

total$=$ 2

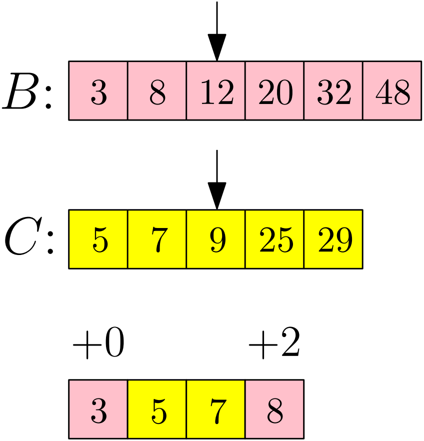
C: 5 | 7 | 9 | 25 | 29

$+0$ $+2$

3 | 5 | 7 | 8

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B:

3	8	12	20	32	48

C:

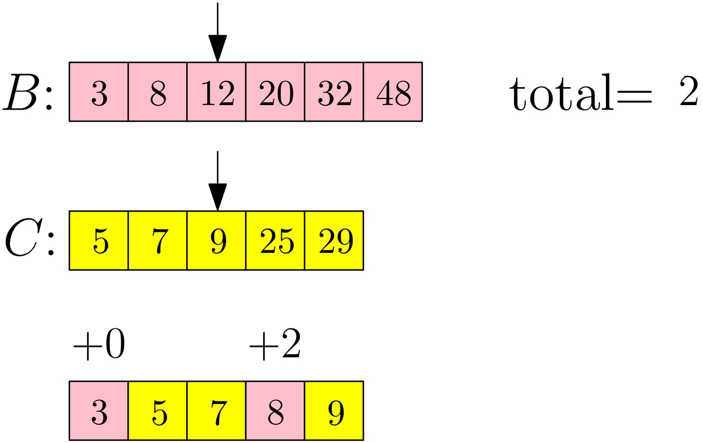
5	7	9	25	29

+0			+2
3	5	7	8

total$=$ 2

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B: 3 | 8 | 12 | 20 | 32 | 48

total= 2

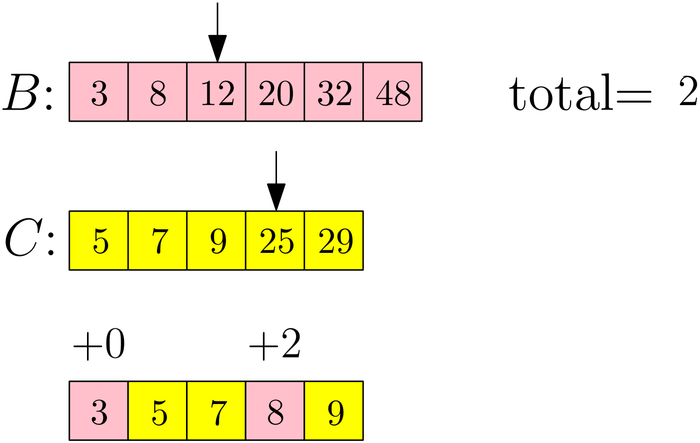
C: 5 | 7 | 9 | 25 | 29

$+0$ $+2$

3 | 5 | 7 | 8 | 9

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B: 3 | 8 | 12 | 20 | 32 | 48

total $= 2$

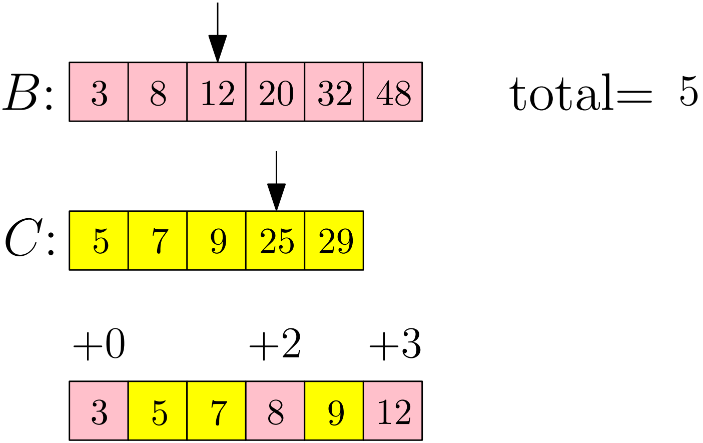
C: 5 | 7 | 9 | 25 | 29

$+0$ $+2$

3 | 5 | 7 | 8 | 9

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B: | 3 | 8 | 12 | 20 | 32 | 48 |

total$=$ 5

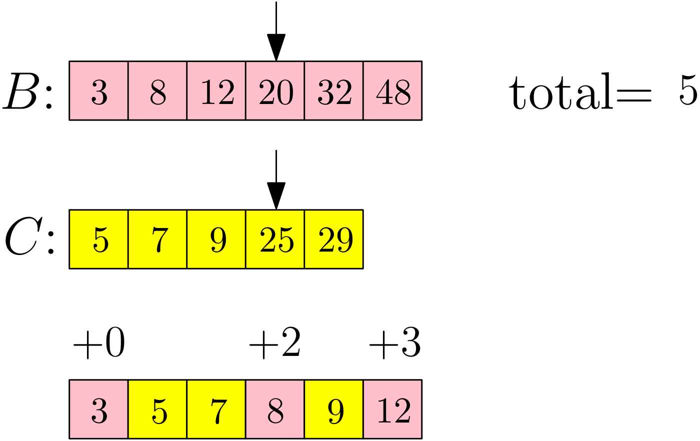
C: | 5 | 7 | 9 | 25 | 29 |

$+0$ $+2$ $+3$

| 3 | 5 | 7 | 8 | 9 | 12 |

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B:

3	8	12	20	32	48

total$=$ 5

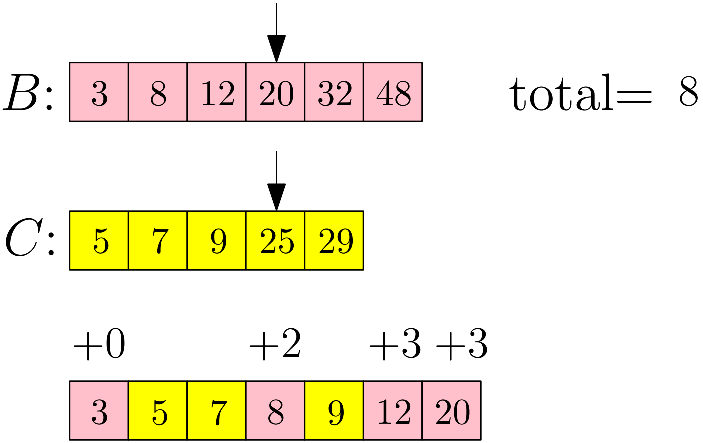
C:

5	7	9	25	29

+0			+2		+3
3	5	7	8	9	12

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B:

3	8	12	20	32	48

total= 8

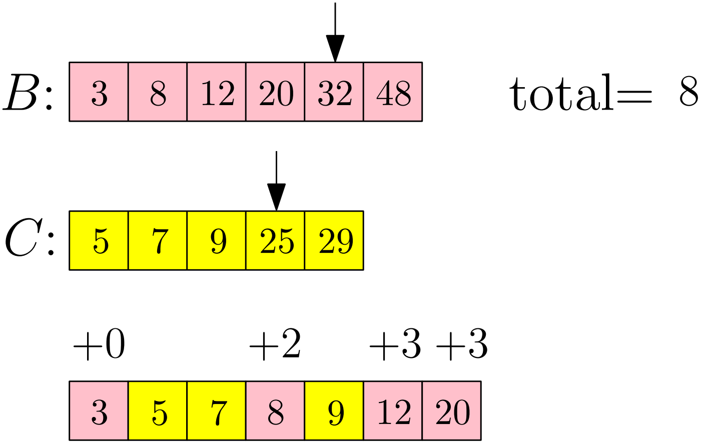
C:

5	7	9	25	29

+0			+2		+3	+3
3	5	7	8	9	12	20

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B:	3	8	12	20	32	48

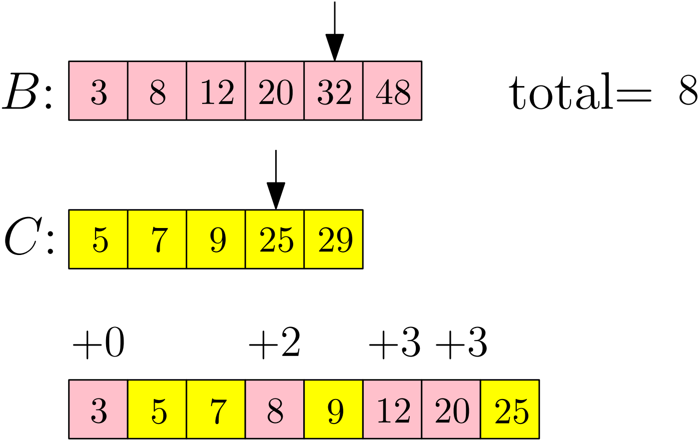
total$=$ 8

C:	5	7	9	25	29

$+0$			$+2$		$+3$	$+3$
3	5	7	8	9	12	20

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B: 3 | 8 | 12 | 20 | 32 | 48

total= 8

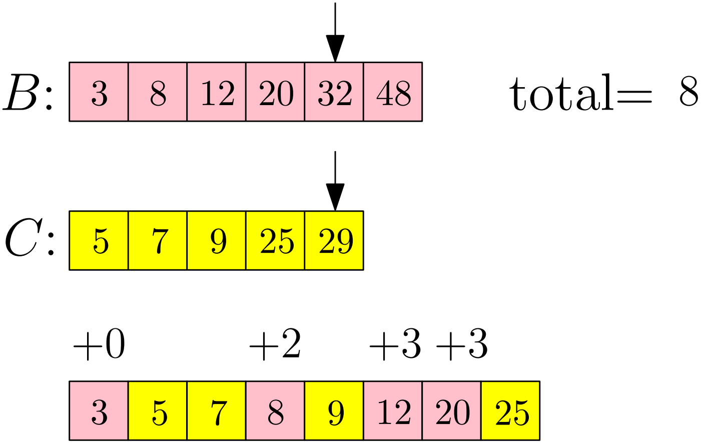
C: 5 | 7 | 9 | 25 | 29

+0 +2 +3 +3

3 | 5 | 7 | 8 | 9 | 12 | 20 | 25

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B: 3 | 8 | 12 | 20 | 32 | 48

total= 8

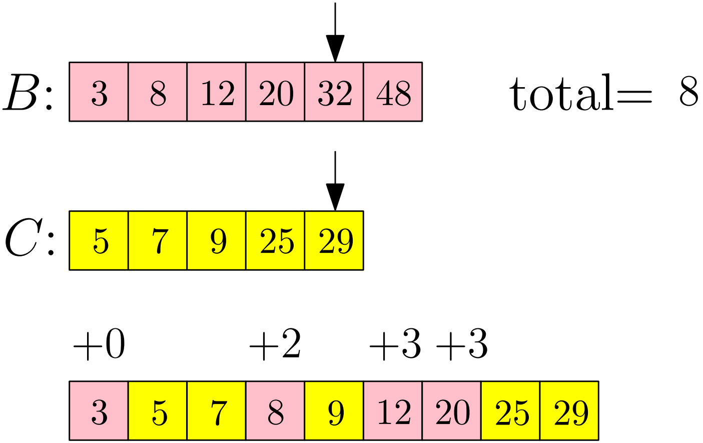
C: 5 | 7 | 9 | 25 | 29

+0 +2 +3 +3

3 | 5 | 7 | 8 | 9 | 12 | 20 | 25

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B:

3	8	12	20	32	48

total= 8

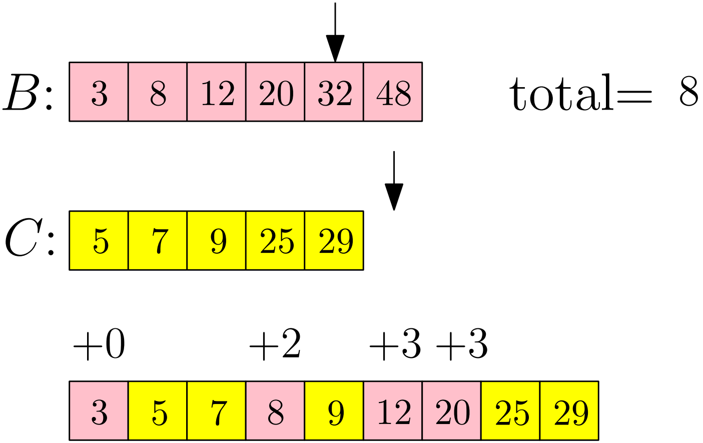
C:

5	7	9	25	29

+0			+2		+3	+3		
3	5	7	8	9	12	20	25	29

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B:

3	8	12	20	32	48

total$=$ 8

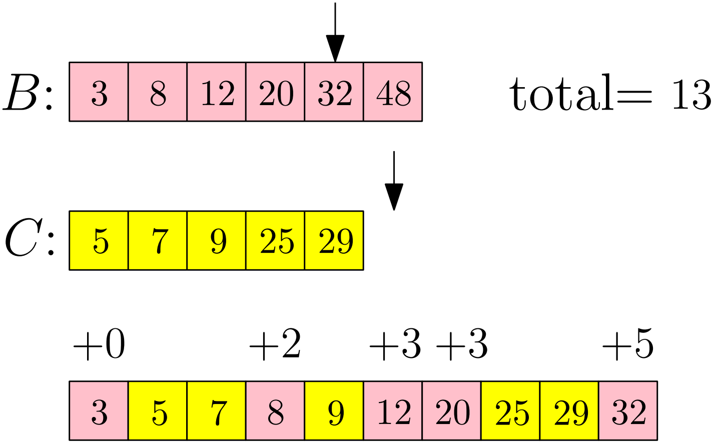
C:

5	7	9	25	29

$+0$			$+2$		$+3$	$+3$		
3	5	7	8	9	12	20	25	29

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B: 3 8 12 20 32 48

total= 13

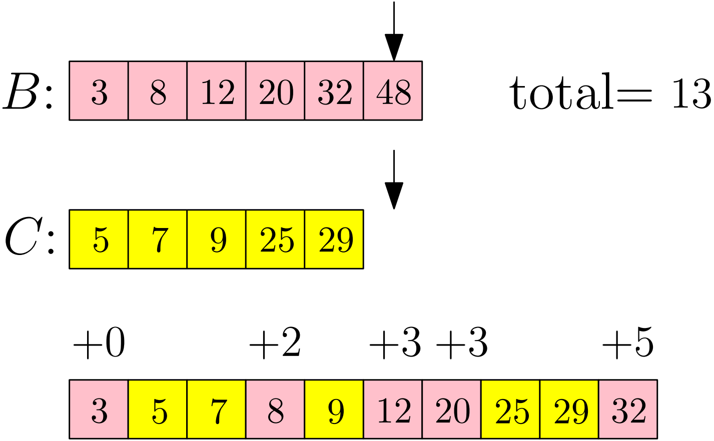
C: 5 7 9 25 29

+0 +2 +3 +3 +5

3 5 7 8 9 12 20 25 29 32

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B:

3	8	12	20	32	48

total $= 13$

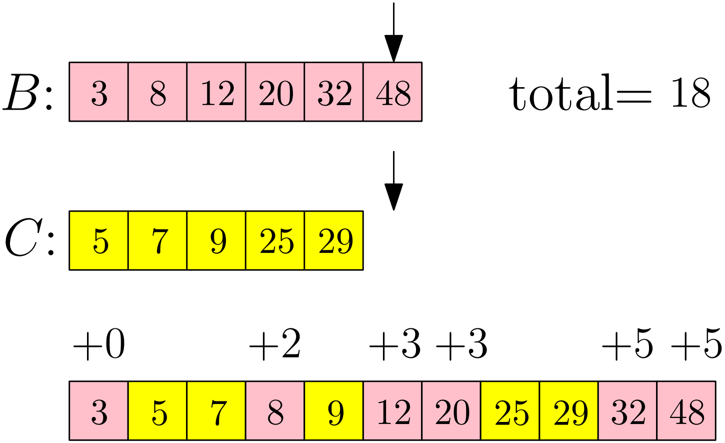
C:

5	7	9	25	29

+0			+2		+3	+3			+5
3	5	7	8	9	12	20	25	29	32

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B:	3	8	12	20	32	48

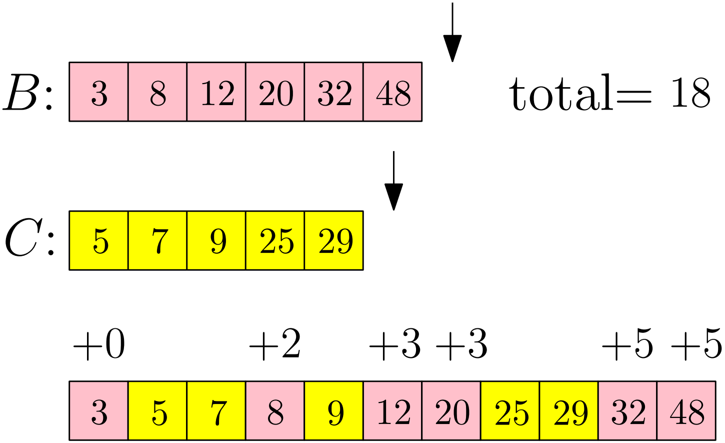
$\text{total} = 18$

C:	5	7	9	25	29

$+0$			$+2$		$+3$	$+3$			$+5$	$+5$
3	5	7	8	9	12	20	25	29	32	48

Counting Inversions between B and C

Count pairs i, j such that $B[i] > C[j]$:



B: 3 8 12 20 32 48

total= 18

C: 5 7 9 25 29

$+0$ $+2$ $+3$ $+3$ $+5$ $+5$

3 5 7 8 9 12 20 25 29 32 48

Count Inversions between B and C

- Procedure that merges B and C and counts inversions between B and C at the same time

merge-and-count(B, C, n_1, n_2)

```
count ← 0;
A ← array of size $n_1 + n_2$; $i \leftarrow 1$; $j \leftarrow 1$
while $i \le n_1$ or $j \le n_2$ do
    if $j > n_2$ or ($i \le n_1$ and $B[i] \le C[j]$) then
        $A[i+j-1] \leftarrow B[i]$; $i \leftarrow i+1$
        $count \leftarrow count + (j-1)$
    else
        $A[i+j-1] \leftarrow C[j]$; $j \leftarrow j+1$
return $(A, count)$
```

Sort and Count Inversions in A

- A procedure that returns the sorted array of A and counts the number of inversions in A:

sort-and-count(A, n)

```
if $n = 1$ then
    return $(A, 0)$
else
    $(B, m_1) \leftarrow$ sort-and-count$\left(A\left[1..\lfloor n/2 \rfloor\right], \lfloor n/2 \rfloor\right)$
    $(C, m_2) \leftarrow$ sort-and-count$\left(A\left[\lfloor n/2 \rfloor + 1..n\right], \lceil n/2 \rceil\right)$
    $(A, m_3) \leftarrow$ merge-and-count$(B, C, \lfloor n/2 \rfloor, \lceil n/2 \rceil)$
    return $(A, m_1 + m_2 + m_3)$
```

Sort and Count Inversions in A

- A procedure that returns the sorted array of A and counts the number of inversions in A:

sort-and-count(A, n)

```
1: if $n = 1$ then
2:     return $(A, 0)$
3: else
4:     $(B, m_1) \leftarrow$ sort-and-count$\left(A\left[1..\lfloor n/2 \rfloor\right], \lfloor n/2 \rfloor\right)$
5:     $(C, m_2) \leftarrow$ sort-and-count$\left(A\left[\lfloor n/2 \rfloor + 1..n\right], \lceil n/2 \rceil\right)$
6:     $(A, m_3) \leftarrow$ merge-and-count$(B, C, \lfloor n/2 \rfloor, \lceil n/2 \rceil)$
7:     return $(A, m_1 + m_2 + m_3)$
```

- Divide: trivial
- Conquer: 4, 5
- Combine: 6, 7

sort-and-count(A, n)

```
if $n = 1$ then
    return $(A, 0)$
else
    $(B, m_1) \leftarrow$ sort-and-count$\Big(A\big[1..\lfloor n/2\rfloor\big], \lfloor n/2\rfloor\Big)$
    $(C, m_2) \leftarrow$ sort-and-count$\Big(A\big[\lfloor n/2\rfloor + 1..n\big], \lceil n/2\rceil\Big)$
    $(A, m_3) \leftarrow$ merge-and-count$(B, C, \lfloor n/2\rfloor, \lceil n/2\rceil)$
    return $(A, m_1 + m_2 + m_3)$
```

- Recurrence for the running time: $T(n) = 2T(n/2) + O(n)$

sort-and-count(A, n)

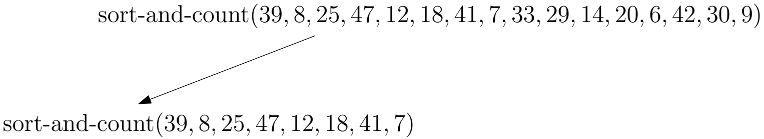
```
if $n = 1$ then
    return $(A, 0)$
else
    $(B, m_1) \leftarrow$ sort-and-count$\Big(A\big[1..\lfloor n/2\rfloor\big], \lfloor n/2\rfloor\Big)$
    $(C, m_2) \leftarrow$ sort-and-count$\Big(A\big[\lfloor n/2\rfloor + 1..n\big], \lceil n/2\rceil\Big)$
    $(A, m_3) \leftarrow$ merge-and-count$(B, C, \lfloor n/2\rfloor, \lceil n/2\rceil)$
    return $(A, m_1 + m_2 + m_3)$
```

- Recurrence for the running time: $T(n) = 2T(n/2) + O(n)$
- Running time $= O(n \log n)$

Example

$$\text{sort-and-count}(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$$

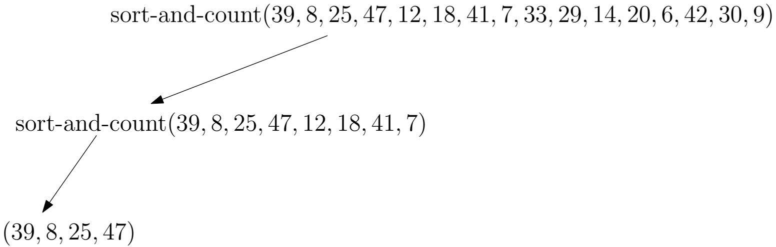
Example



sort-and-count(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)

sort-and-count(39, 8, 25, 47, 12, 18, 41, 7)

Example

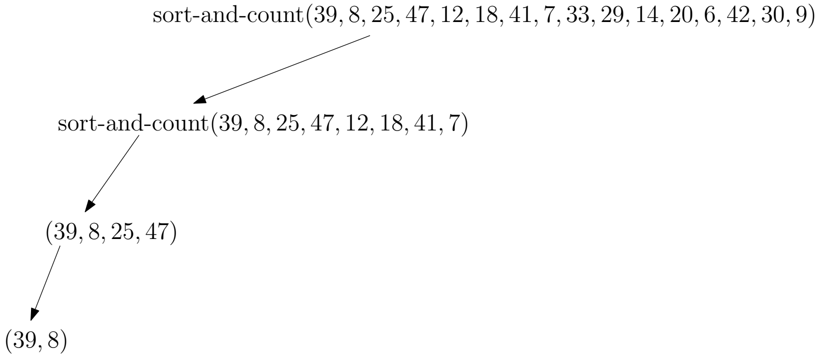


sort-and-count(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)

sort-and-count(39, 8, 25, 47, 12, 18, 41, 7)

(39, 8, 25, 47)

Example



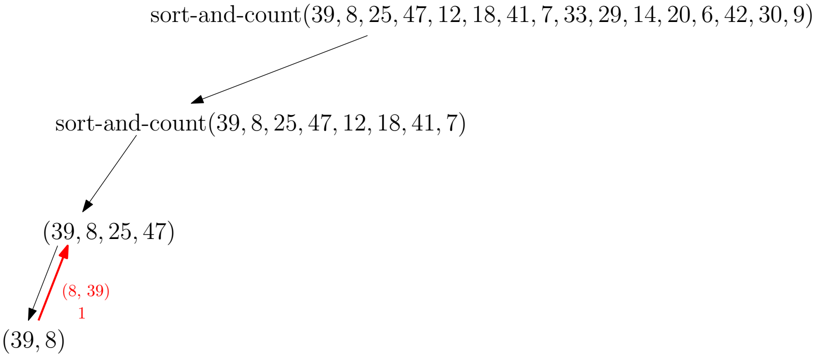
sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$

sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7)$

$(39, 8, 25, 47)$

$(39, 8)$

Example



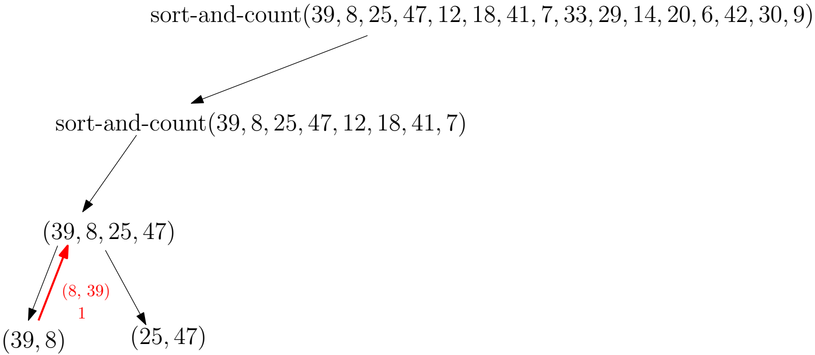
sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$

sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7)$

$(39, 8, 25, 47)$

$(39, 8)$

Example



sort-and-count(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)

sort-and-count(39, 8, 25, 47, 12, 18, 41, 7)

(39, 8, 25, 47)

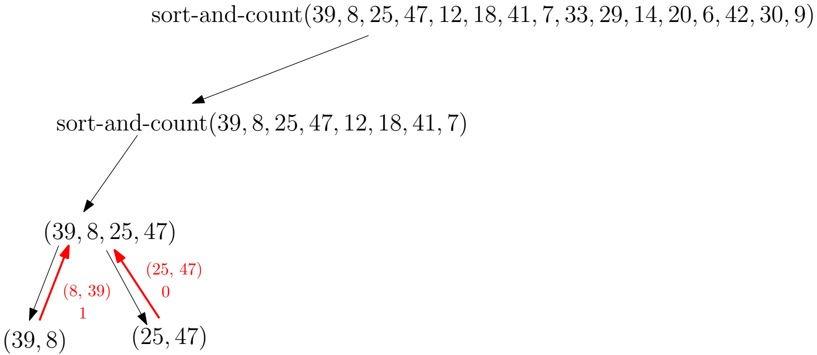
(39, 8)

(25, 47)

(8, 39)

1

Example



sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$

sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7)$

$(39, 8, 25, 47)$

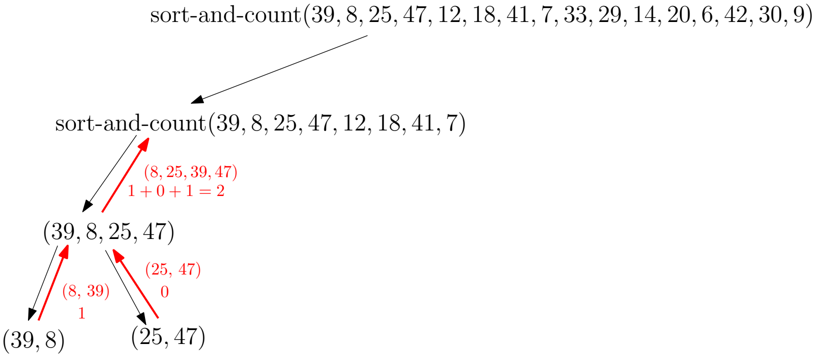
(8, 39)
1

(25, 47)
0

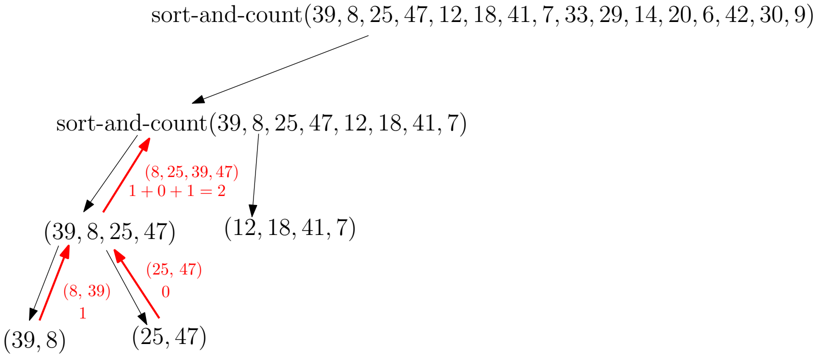
$(39, 8)$

$(25, 47)$

Example


sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$
sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7)$
$(8, 25, 39, 47)$
$1 + 0 + 1 = 2$
$(39, 8, 25, 47)$
$(8,\ 39)$
1
$(25,\ 47)$
0
$(39, 8)$
$(25, 47)$

Example



sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$

sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7)$

$(8, 25, 39, 47)$

$1 + 0 + 1 = 2$

$(39, 8, 25, 47)$

$(12, 18, 41, 7)$

$(25,\ 47)$

0

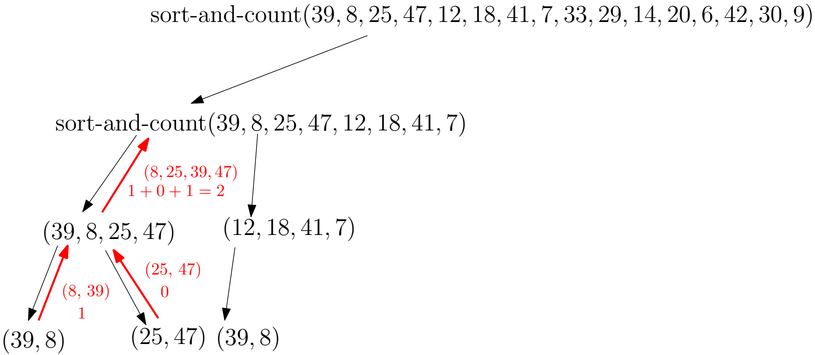
$(8,\ 39)$

1

$(39, 8)$

$(25, 47)$

Example



sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$

sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7)$

$(8, 25, 39, 47)$

$1 + 0 + 1 = 2$

$(39, 8, 25, 47)$

$(12, 18, 41, 7)$

$(8,\ 39)$

1

$(25,\ 47)$

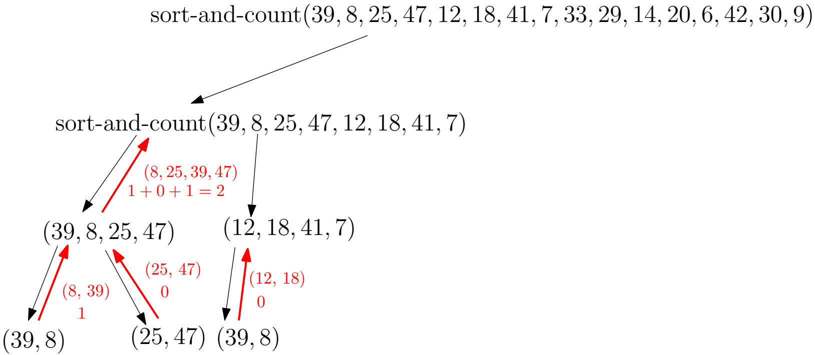
0

$(39, 8)$

$(25, 47)$

$(39, 8)$

Example



sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$

sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7)$

$(8, 25, 39, 47)$

$1 + 0 + 1 = 2$

$(39, 8, 25, 47)$ $(12, 18, 41, 7)$

$(8,\ 39)$

1

$(25,\ 47)$

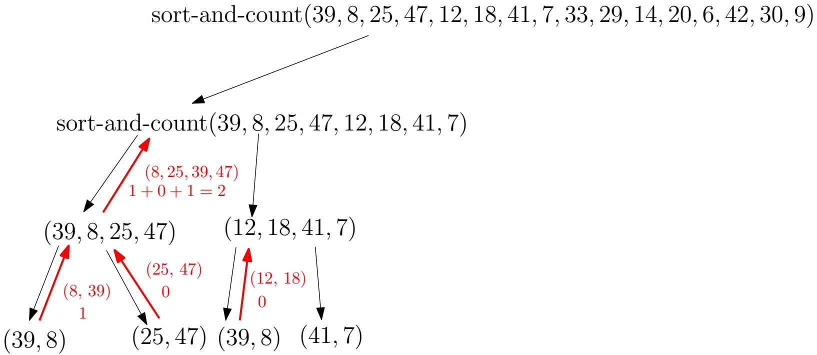
0

$(12,\ 18)$

0

$(39, 8)$ $(25, 47)$ $(39, 8)$

Example



sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$

sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7)$

$(8, 25, 39, 47)$

$1 + 0 + 1 = 2$

$(39, 8, 25, 47)$

$(12, 18, 41, 7)$

$(8,\ 39)$

1

$(25,\ 47)$

0

$(12,\ 18)$

0

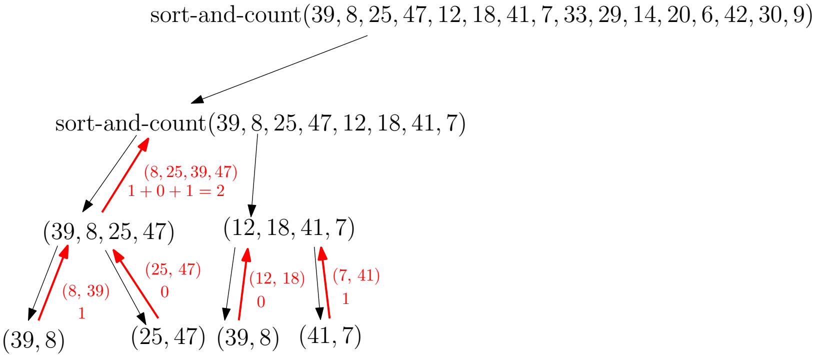
$(39, 8)$

$(25, 47)$

$(39, 8)$

$(41, 7)$

Example



sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$

sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7)$

$(8, 25, 39, 47)$

$1 + 0 + 1 = 2$

$(39, 8, 25, 47)$

$(12, 18, 41, 7)$

(8, 39)

1

(25, 47)

0

(12, 18)

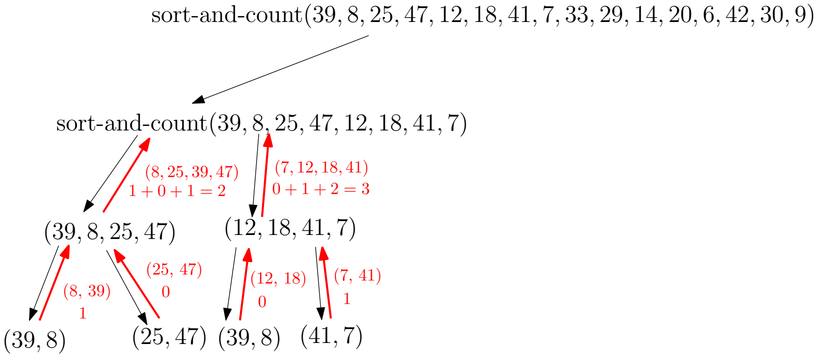
0

(7, 41)

1

$(39, 8)$ $(25, 47)$ $(39, 8)$ $(41, 7)$

Example



sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$

sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7)$

$(8, 25, 39, 47)$

$1 + 0 + 1 = 2$

$(7, 12, 18, 41)$

$0 + 1 + 2 = 3$

$(39, 8, 25, 47)$

$(12, 18, 41, 7)$

$(25,\ 47)$

0

$(8,\ 39)$

1

$(12,\ 18)$

0

$(7,\ 41)$

1

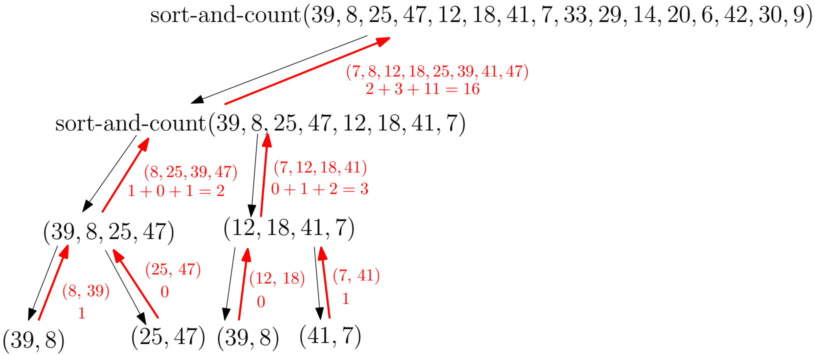
$(39, 8)$

$(25, 47)$

$(39, 8)$

$(41, 7)$

Example



sort-and-count(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)

(7, 8, 12, 18, 25, 39, 41, 47)
$2 + 3 + 11 = 16$

sort-and-count(39, 8, 25, 47, 12, 18, 41, 7)

(8, 25, 39, 47)
$1 + 0 + 1 = 2$

(7, 12, 18, 41)
$0 + 1 + 2 = 3$

(39, 8, 25, 47)

(12, 18, 41, 7)

(8, 39)
1

(25, 47)
0

(12, 18)
0

(7, 41)
1

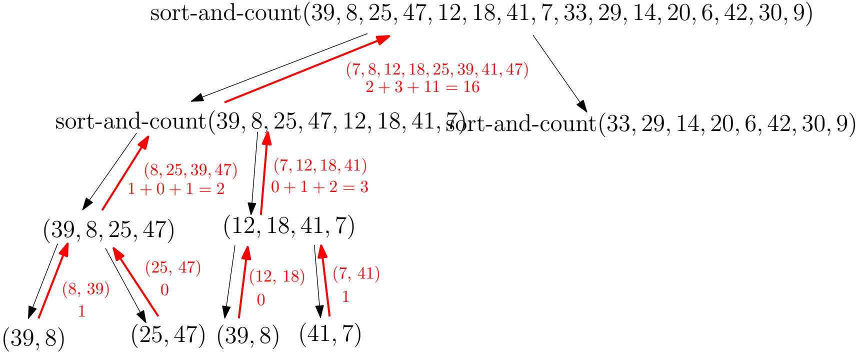
(39, 8)

(25, 47)

(39, 8)

(41, 7)

Example



sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$

$(7, 8, 12, 18, 25, 39, 41, 47)$
$2 + 3 + 11 = 16$

sort-and-count$(39, 8, 25, 47, 12, 18, 41, 7)$

sort-and-count$(33, 29, 14, 20, 6, 42, 30, 9)$

$(8, 25, 39, 47)$
$1 + 0 + 1 = 2$

$(7, 12, 18, 41)$
$0 + 1 + 2 = 3$

$(39, 8, 25, 47)$

$(12, 18, 41, 7)$

$(8,\ 39)$
1

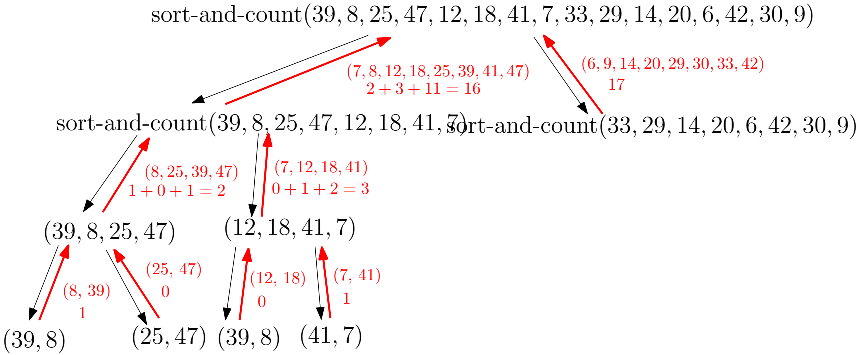
$(25,\ 47)$
0

$(12,\ 18)$
0

$(7,\ 41)$
1

$(39, 8)$ $(25, 47)$ $(39, 8)$ $(41, 7)$

Example



$\text{sort-and-count}(39, 8, 25, 47, 12, 18, 41, 7, 33, 29, 14, 20, 6, 42, 30, 9)$

$(7, 8, 12, 18, 25, 39, 41, 47)$

$2 + 3 + 11 = 16$

$(6, 9, 14, 20, 29, 30, 33, 42)$

17

$\text{sort-and-count}(39, 8, 25, 47, 12, 18, 41, 7)$

$\text{sort-and-count}(33, 29, 14, 20, 6, 42, 30, 9)$

$(8, 25, 39, 47)$

$1 + 0 + 1 = 2$

$(7, 12, 18, 41)$

$0 + 1 + 2 = 3$

$(39, 8, 25, 47)$

$(12, 18, 41, 7)$

$(8,\ 39)$

1

$(25,\ 47)$

0

$(12,\ 18)$

0

$(7,\ 41)$

1

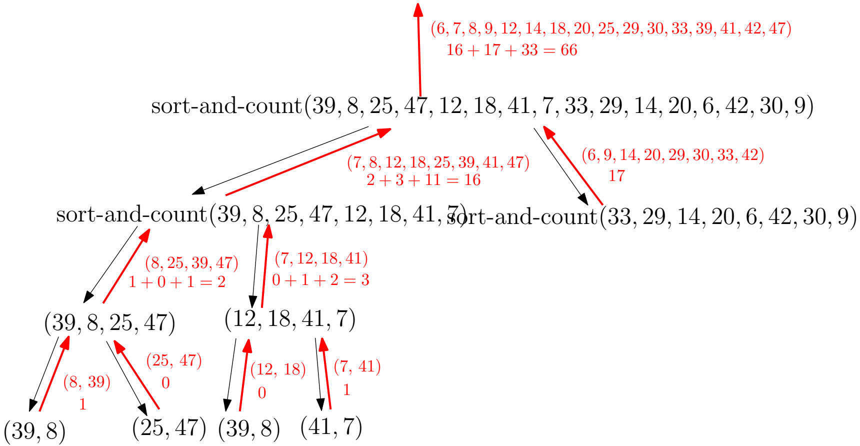
$(39, 8)$

$(25, 47)$

$(39, 8)$

$(41, 7)$

Example



$(6,7,8,9,12,14,18,20,25,29,30,33,39,41,42,47)$

$16+17+33=66$

sort-and-count$(39,8,25,47,12,18,41,7,33,29,14,20,6,42,30,9)$

$(7,8,12,18,25,39,41,47)$

$2+3+11=16$

$(6,9,14,20,29,30,33,42)$

17

sort-and-count$(39,8,25,47,12,18,41,7)$

sort-and-count$(33,29,14,20,6,42,30,9)$

$(8,25,39,47)$

$1+0+1=2$

$(7,12,18,41)$

$0+1+2=3$

$(39,8,25,47)$

$(12,18,41,7)$

$(8,\ 39)$

1

$(25,\ 47)$

0

$(12,\ 18)$

0

$(7,\ 41)$

1

$(39,8)$ $(25,47)$ $(39,8)$ $(41,7)$

Outline

1. Divide-and-Conquer
2. Counting Inversions
3. Solving Recurrences
4. Quicksort and Selection
    - Quicksort
    - Lower Bound for Comparison-Based Sorting Algorithms
    - Selection Problem
5. Polynomial Multiplication
6. Strassen's Algorithm for Matrix Multiplication
7. FFT(Fast Fourier Transform): Polynomial Multiplication in $O(n\log n)$ Time
8. Finding Closest Pair of Points in 2D Euclidean Space
9. Computing n-th Fibonacci Number

Methods for Solving Recurrences

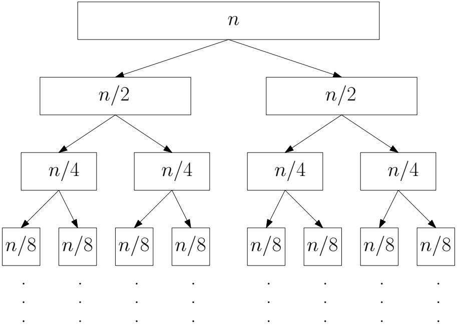
- The recursion-tree method
- The master theorem

Recursion-Tree Method

- $T(n) = 2T(n/2) + O(n)$

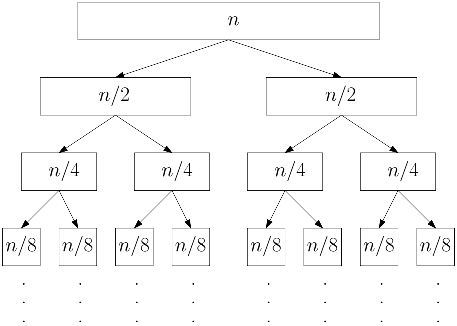
Recursion-Tree Method

- $T(n) = 2T(n/2) + O(n)$



Recursion-Tree Method

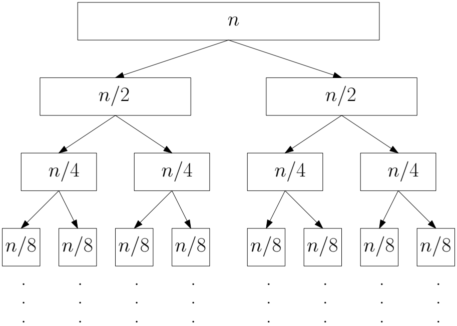
- $T(n) = 2T(n/2) + O(n)$



- Each level takes running time $O(n)$

Recursion-Tree Method

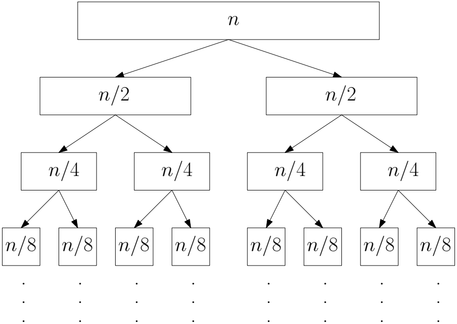
- $T(n) = 2T(n/2) + O(n)$



- Each level takes running time $O(n)$
- There are $O(\log n)$ levels

Recursion-Tree Method

- $T(n) = 2T(n/2) + O(n)$



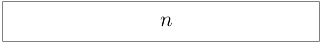
- Each level takes running time $O(n)$
- There are $O(\log n)$ levels
- Running time $= O(n \log n)$

Recursion-Tree Method

- $T(n) = 3T(n/2) + O(n)$

Recursion-Tree Method

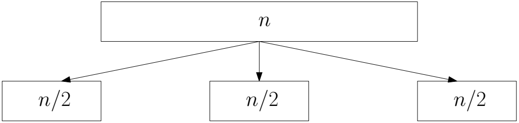
- $T(n) = 3T(n/2) + O(n)$



n

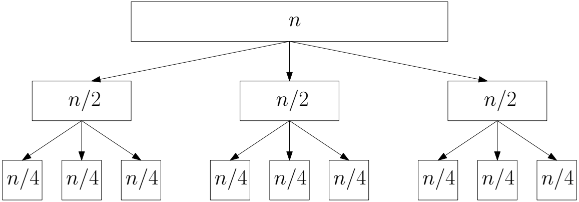
Recursion-Tree Method

- $T(n) = 3T(n/2) + O(n)$



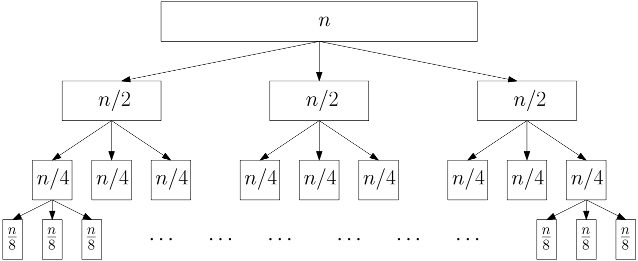
Recursion-Tree Method

- $T(n) = 3T(n/2) + O(n)$



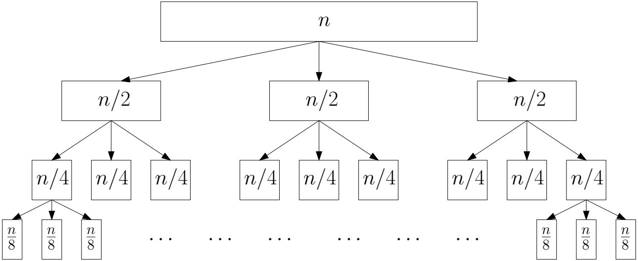
Recursion-Tree Method

- $T(n) = 3T(n/2) + O(n)$



Recursion-Tree Method

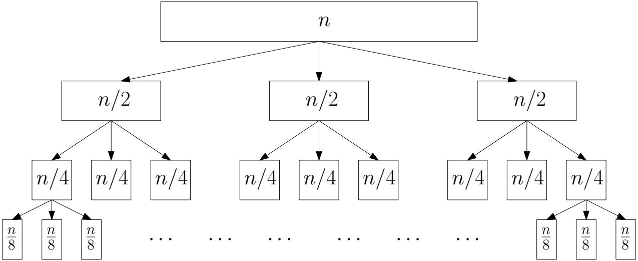
- $T(n) = 3T(n/2) + O(n)$



- Total running time at level i?

Recursion-Tree Method

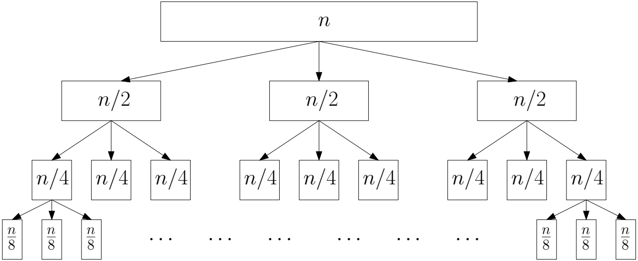
- $T(n) = 3T(n/2) + O(n)$



- Total running time at level i? $\frac{n}{2^i} \times 3^i = \left(\frac{3}{2}\right)^i n$

Recursion-Tree Method

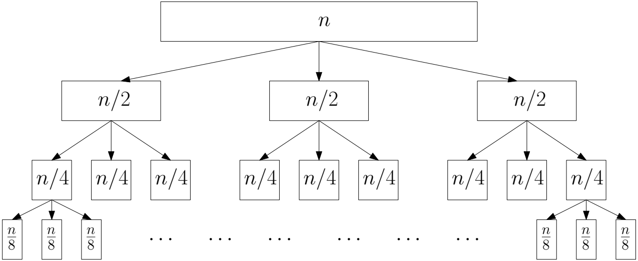
- $T(n) = 3T(n/2) + O(n)$



- Total running time at level i? $\frac{n}{2^i} \times 3^i = \left(\frac{3}{2}\right)^i n$
- Index of last level?

Recursion-Tree Method

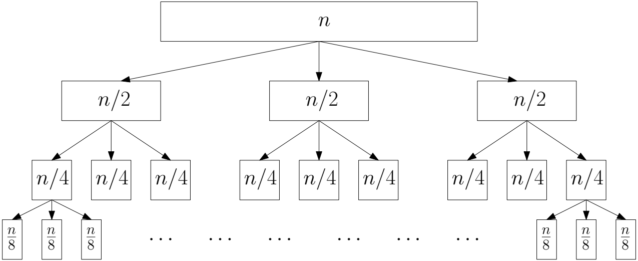
- $T(n) = 3T(n/2) + O(n)$



- Total running time at level i? $\frac{n}{2^i} \times 3^i = \left(\frac{3}{2}\right)^i n$
- Index of last level? $\log_2 n$

Recursion-Tree Method

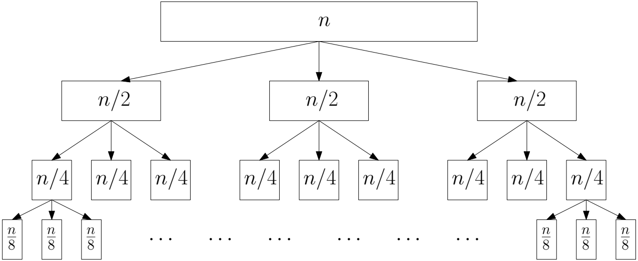
- $T(n) = 3T(n/2) + O(n)$



- Total running time at level i? $\frac{n}{2^i} \times 3^i = \left(\frac{3}{2}\right)^i n$
- Index of last level? $\log_2 n$
- Total running time?

Recursion-Tree Method

- $T(n) = 3T(n/2) + O(n)$



- Total running time at level i? $\frac{n}{2^i} \times 3^i = \left(\frac{3}{2}\right)^i n$
- Index of last level? $\log_2 n$
- Total running time?

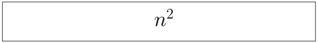
$$\sum_{i=0}^{\log_2 n} \left(\frac{3}{2}\right)^i n = O\left(n\left(\frac{3}{2}\right)^{\log_2 n}\right) = O(3^{\log_2 n}) = O(n^{\log_2 3}).$$

Recursion-Tree Method

- $T(n) = 3T(n/2) + O(n^2)$

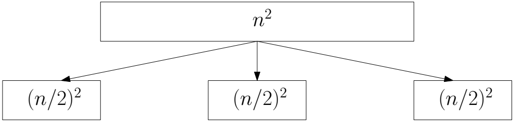
Recursion-Tree Method

- $T(n) = 3T(n/2) + O(n^2)$



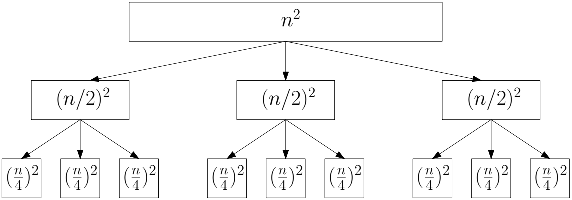
Recursion-Tree Method

- $T(n) = 3T(n/2) + O(n^2)$



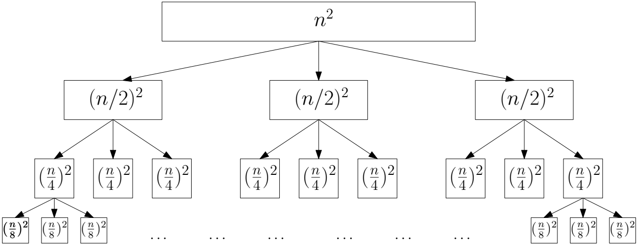
Recursion-Tree Method

- $T(n) = 3T(n/2) + O(n^2)$



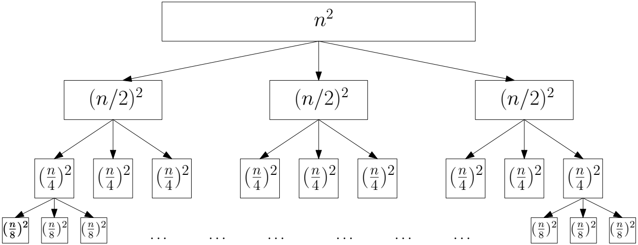
Recursion-Tree Method

- $T(n) = 3T(n/2) + O(n^2)$



Recursion-Tree Method

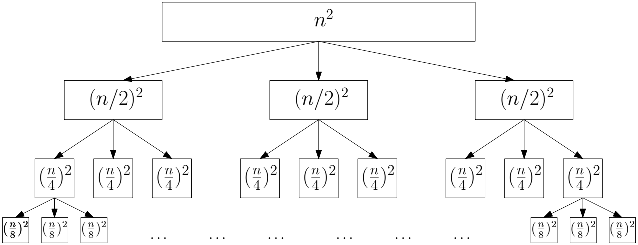
- $T(n) = 3T(n/2) + O(n^2)$



- Total running time at level i?

Recursion-Tree Method

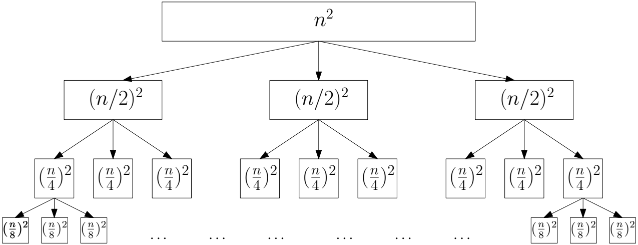
- $T(n) = 3T(n/2) + O(n^2)$



- Total running time at level i? $\left(\frac{n}{2^i}\right)^2 \times 3^i = \left(\frac{3}{4}\right)^i n^2$

Recursion-Tree Method

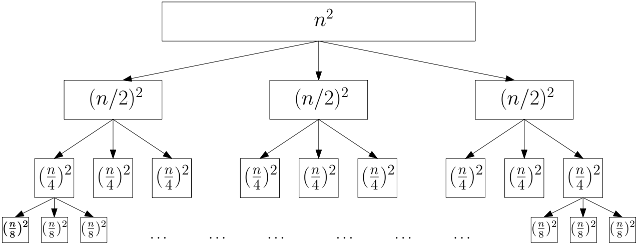
- $T(n) = 3T(n/2) + O(n^2)$



- Total running time at level i? $\left(\frac{n}{2^i}\right)^2 \times 3^i = \left(\frac{3}{4}\right)^i n^2$
- Index of last level?

Recursion-Tree Method

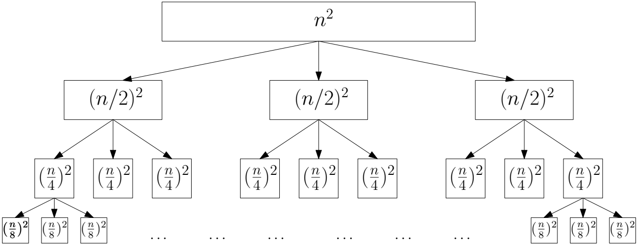
- $T(n) = 3T(n/2) + O(n^2)$



- Total running time at level i? $\left(\frac{n}{2^i}\right)^2 \times 3^i = \left(\frac{3}{4}\right)^i n^2$
- Index of last level? $\log_2 n$

Recursion-Tree Method

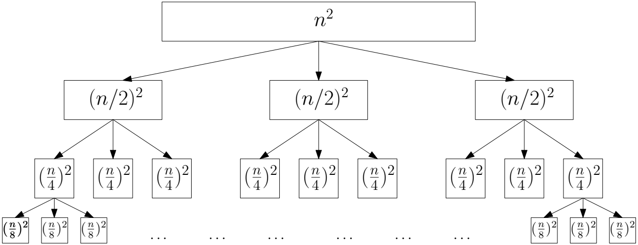
- $T(n) = 3T(n/2) + O(n^2)$



- Total running time at level i? $\left(\frac{n}{2^i}\right)^2 \times 3^i = \left(\frac{3}{4}\right)^i n^2$
- Index of last level? $\log_2 n$
- Total running time?

Recursion-Tree Method

- $T(n) = 3T(n/2) + O(n^2)$

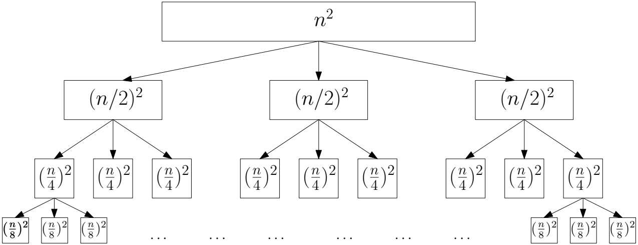


- Total running time at level i? $\left(\frac{n}{2^i}\right)^2 \times 3^i = \left(\frac{3}{4}\right)^i n^2$
- Index of last level? $\log_2 n$
- Total running time?

$$\sum_{i=0}^{\log_2 n} \left(\frac{3}{4}\right)^i n^2 =$$

Recursion-Tree Method

- $T(n) = 3T(n/2) + O(n^2)$



- Total running time at level i? $\left(\frac{n}{2^i}\right)^2 \times 3^i = \left(\frac{3}{4}\right)^i n^2$
- Index of last level? $\log_2 n$
- Total running time?

$$\sum_{i=0}^{\log_2 n} \left(\frac{3}{4}\right)^i n^2 = O(n^2).$$

Master Theorem

Recurrences	a	b	c	time
$T(n) = 2T(n/2) + O(n)$				$O(n \log n)$
$T(n) = 3T(n/2) + O(n)$				$O(n^{\log_2 3})$
$T(n) = 3T(n/2) + O(n^2)$				$O(n^2)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

Master Theorem

Recurrences	a	b	c	time
$T(n) = 2T(n/2) + O(n)$	2	2	1	$O(n \log n)$
$T(n) = 3T(n/2) + O(n)$				$O(n^{\log_2 3})$
$T(n) = 3T(n/2) + O(n^2)$				$O(n^2)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

Master Theorem

Recurrences	a	b	c	time
$T(n) = 2T(n/2) + O(n)$	2	2	1	$O(n \log n)$
$T(n) = 3T(n/2) + O(n)$	3	2	1	$O(n^{\log_2 3})$
$T(n) = 3T(n/2) + O(n^2)$				$O(n^2)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

Master Theorem

Recurrences	a	b	c	time
$T(n) = 2T(n/2) + O(n)$	2	2	1	$O(n \log n)$
$T(n) = 3T(n/2) + O(n)$	3	2	1	$O(n^{\log_2 3})$
$T(n) = 3T(n/2) + O(n^2)$	3	2	2	$O(n^2)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

Master Theorem

Recurrences	a	b	c	time
$T(n) = 2T(n/2) + O(n)$	2	2	1	$O(n \log n)$
$T(n) = 3T(n/2) + O(n)$	3	2	1	$O(n^{\log_2 3})$
$T(n) = 3T(n/2) + O(n^2)$	3	2	2	$O(n^2)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} & \text{if } c < \log_b a \\ & \text{if } c = \log_b a \\ & \text{if } c > \log_b a \end{cases}$$

Master Theorem

Recurrences	a	b	c	time
$T(n) = 2T(n/2) + O(n)$	2	2	1	$O(n \log n)$
$T(n) = 3T(n/2) + O(n)$	3	2	1	$O(n^{\log_2 3})$
$T(n) = 3T(n/2) + O(n^2)$	3	2	2	$O(n^2)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} ?? & \text{if } c < \log_b a \\ & \text{if } c = \log_b a \\ & \text{if } c > \log_b a \end{cases}$$

Master Theorem

Recurrences	a	b	c	time
$T(n) = 2T(n/2) + O(n)$	2	2	1	$O(n \log n)$
$T(n) = 3T(n/2) + O(n)$	3	2	1	$O(n^{\log_2 3})$
$T(n) = 3T(n/2) + O(n^2)$	3	2	2	$O(n^2)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ & \text{if } c = \log_b a \\ & \text{if } c > \log_b a \end{cases}$$

Master Theorem

Recurrences	a	b	c	time
$T(n) = 2T(n/2) + O(n)$	2	2	1	$O(n \log n)$
$T(n) = 3T(n/2) + O(n)$	3	2	1	$O(n^{\log_2 3})$
$T(n) = 3T(n/2) + O(n^2)$	3	2	2	$O(n^2)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ & \text{if } c = \log_b a \\ ?? & \text{if } c > \log_b a \end{cases}$$

Master Theorem

Recurrences	a	b	c	time
$T(n) = 2T(n/2) + O(n)$	2	2	1	$O(n \log n)$
$T(n) = 3T(n/2) + O(n)$	3	2	1	$O(n^{\log_2 3})$
$T(n) = 3T(n/2) + O(n^2)$	3	2	2	$O(n^2)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

Master Theorem

Recurrences	a	b	c	time
$T(n) = 2T(n/2) + O(n)$	2	2	1	$O(n \log n)$
$T(n) = 3T(n/2) + O(n)$	3	2	1	$O(n^{\log_2 3})$
$T(n) = 3T(n/2) + O(n^2)$	3	2	2	$O(n^2)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ ?? & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

Master Theorem

Recurrences	a	b	c	time
$T(n) = 2T(n/2) + O(n)$	2	2	1	$O(n \log n)$
$T(n) = 3T(n/2) + O(n)$	3	2	1	$O(n^{\log_2 3})$
$T(n) = 3T(n/2) + O(n^2)$	3	2	2	$O(n^2)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

- Ex: $T(n) = 4T(n/2) + O(n^2)$. Which Case?

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

- Ex: $T(n) = 4T(n/2) + O(n^2)$. Case 2.

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

- Ex: $T(n) = 4T(n/2) + O(n^2)$. Case 2. $T(n) = O(n^2 \log n)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

- Ex: $T(n) = 4T(n/2) + O(n^2)$. Case 2. $T(n) = O(n^2 \log n)$
- Ex: $T(n) = 3T(n/2) + O(n)$. Which Case?

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

- Ex: $T(n) = 4T(n/2) + O(n^2)$. Case 2. $T(n) = O(n^2 \log n)$
- Ex: $T(n) = 3T(n/2) + O(n)$. Case 1.

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

- Ex: $T(n) = 4T(n/2) + O(n^2)$. Case 2. $T(n) = O(n^2 \log n)$
- Ex: $T(n) = 3T(n/2) + O(n)$. Case 1. $T(n) = O(n^{\log_2 3})$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

- Ex: $T(n) = 4T(n/2) + O(n^2)$. Case 2. $T(n) = O(n^2 \log n)$
- Ex: $T(n) = 3T(n/2) + O(n)$. Case 1. $T(n) = O(n^{\log_2 3})$
- Ex: $T(n) = T(n/2) + O(1)$. Which Case?

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

- Ex: $T(n) = 4T(n/2) + O(n^2)$. Case 2. $T(n) = O(n^2 \log n)$
- Ex: $T(n) = 3T(n/2) + O(n)$. Case 1. $T(n) = O(n^{\log_2 3})$
- Ex: $T(n) = T(n/2) + O(1)$. Case 2.

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

- Ex: $T(n) = 4T(n/2) + O(n^2)$. Case 2. $T(n) = O(n^2 \log n)$
- Ex: $T(n) = 3T(n/2) + O(n)$. Case 1. $T(n) = O(n^{\log_2 3})$
- Ex: $T(n) = T(n/2) + O(1)$. Case 2. $T(n) = O(\log n)$

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

- Ex: $T(n) = 4T(n/2) + O(n^2)$. Case 2. $T(n) = O(n^2 \log n)$
- Ex: $T(n) = 3T(n/2) + O(n)$. Case 1. $T(n) = O(n^{\log_2 3})$
- Ex: $T(n) = T(n/2) + O(1)$. Case 2. $T(n) = O(\log n)$
- Ex: $T(n) = 2T(n/2) + O(n^2)$. Which Case?

Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

- Ex: $T(n) = 4T(n/2) + O(n^2)$. Case 2. $T(n) = O(n^2 \log n)$
- Ex: $T(n) = 3T(n/2) + O(n)$. Case 1. $T(n) = O(n^{\log_2 3})$
- Ex: $T(n) = T(n/2) + O(1)$. Case 2. $T(n) = O(\log n)$
- Ex: $T(n) = 2T(n/2) + O(n^2)$. Case 3.

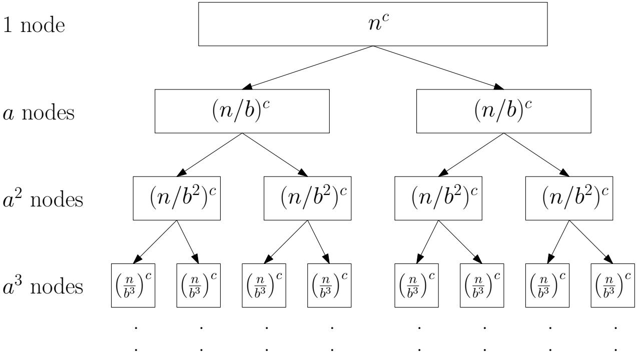
Theorem $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1, b > 1, c \geq 0$ are constants. Then,

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } c < \log_b a \\ O(n^c \log n) & \text{if } c = \log_b a \\ O(n^c) & \text{if } c > \log_b a \end{cases}$$

- Ex: $T(n) = 4T(n/2) + O(n^2)$. Case 2. $T(n) = O(n^2 \log n)$
- Ex: $T(n) = 3T(n/2) + O(n)$. Case 1. $T(n) = O(n^{\log_2 3})$
- Ex: $T(n) = T(n/2) + O(1)$. Case 2. $T(n) = O(\log n)$
- Ex: $T(n) = 2T(n/2) + O(n^2)$. Case 3. $T(n) = O(n^2)$

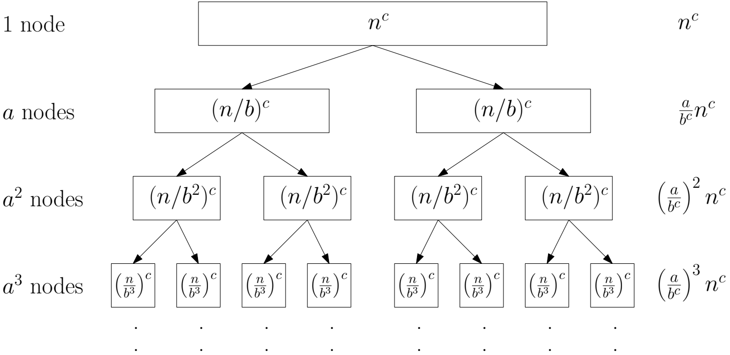
Proof of Master Theorem Using Recursion Tree

$$T(n) = aT(n/b) + O(n^c)$$


1 node
n^c
a nodes
$(n/b)^c$
$(n/b)^c$
a^2 nodes
$(n/b^2)^c$
$(n/b^2)^c$
$(n/b^2)^c$
$(n/b^2)^c$
a^3 nodes
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$

Proof of Master Theorem Using Recursion Tree

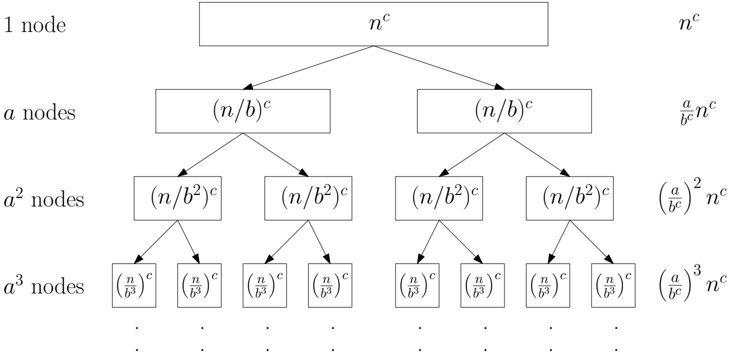
$$T(n) = aT(n/b) + O(n^c)$$



1 node
n^c
n^c
a nodes
$(n/b)^c$
$(n/b)^c$
$\frac{a}{b^c}n^c$
a^2 nodes
$(n/b^2)^c$
$(n/b^2)^c$
$(n/b^2)^c$
$(n/b^2)^c$
$\left(\frac{a}{b^c}\right)^2 n^c$
a^3 nodes
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{a}{b^c}\right)^3 n^c$

Proof of Master Theorem Using Recursion Tree

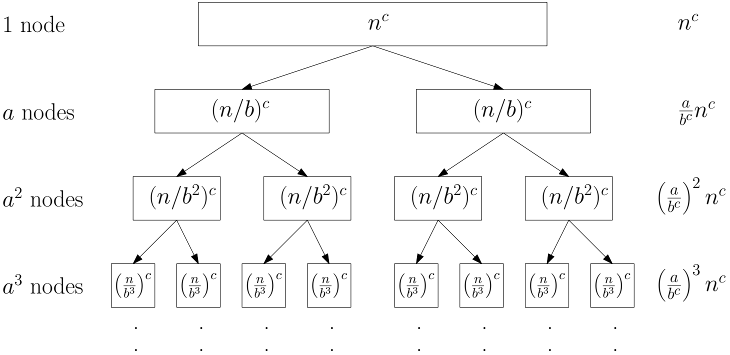
$$T(n) = aT(n/b) + O(n^c)$$


1 node
n^c
n^c
a nodes
$(n/b)^c$
$(n/b)^c$
$\frac{a}{b^c}n^c$
a^2 nodes
$(n/b^2)^c$
$(n/b^2)^c$
$(n/b^2)^c$
$(n/b^2)^c$
$\left(\frac{a}{b^c}\right)^2 n^c$
a^3 nodes
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{a}{b^c}\right)^3 n^c$

- $c < \log_b a$: bottom-level dominates: $\left(\frac{a}{b^c}\right)^{\log_b n} n^c = n^{\log_b a}$

Proof of Master Theorem Using Recursion Tree

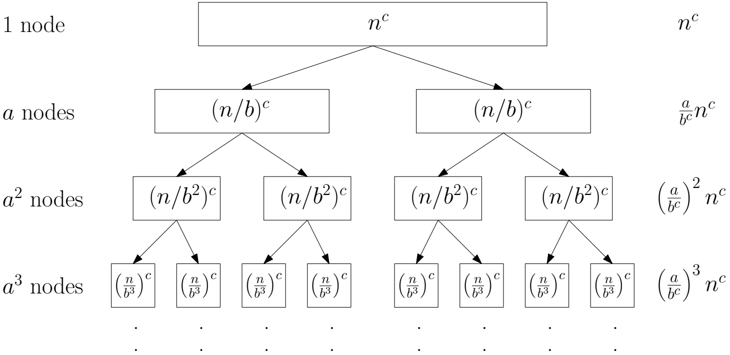
$$T(n) = aT(n/b) + O(n^c)$$



- $c < \log_b a$: bottom-level dominates: $\left(\frac{a}{b^c}\right)^{\log_b n} n^c = n^{\log_b a}$
- $c = \log_b a$: all levels have same time: $n^c \log_b n = O(n^c \log n)$

Proof of Master Theorem Using Recursion Tree

$$T(n) = aT(n/b) + O(n^c)$$


1 node
n^c
n^c
a nodes
$(n/b)^c$
$(n/b)^c$
$\frac{a}{b^c}n^c$
a^2 nodes
$(n/b^2)^c$
$(n/b^2)^c$
$(n/b^2)^c$
$(n/b^2)^c$
$\left(\frac{a}{b^c}\right)^2 n^c$
a^3 nodes
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{n}{b^3}\right)^c$
$\left(\frac{a}{b^c}\right)^3 n^c$

- $c < \log_b a$: bottom-level dominates: $\left(\frac{a}{b^c}\right)^{\log_b n} n^c = n^{\log_b a}$
- $c = \log_b a$: all levels have same time: $n^c \log_b n = O(n^c \log n)$
- $c > \log_b a$: top-level dominates: $O(n^c)$

Outline

1. Divide-and-Conquer
2. Counting Inversions
3. Solving Recurrences
4. Quicksort and Selection
    - Quicksort
    - Lower Bound for Comparison-Based Sorting Algorithms
    - Selection Problem
5. Polynomial Multiplication
6. Strassen's Algorithm for Matrix Multiplication
7. FFT(Fast Fourier Transform): Polynomial Multiplication in $O(n \log n)$ Time
8. Finding Closest Pair of Points in 2D Euclidean Space
9. Computing n-th Fibonacci Number

Outline

1. Divide-and-Conquer
2. Counting Inversions
3. Solving Recurrences
4. Quicksort and Selection
   - Quicksort
   - Lower Bound for Comparison-Based Sorting Algorithms
   - Selection Problem
5. Polynomial Multiplication
6. Strassen's Algorithm for Matrix Multiplication
7. FFT(Fast Fourier Transform): Polynomial Multiplication in $O(n \log n)$ Time
8. Finding Closest Pair of Points in 2D Euclidean Space
9. Computing n-th Fibonacci Number

Quicksort vs Merge-Sort

	Merge Sort	**Quicksort**
Divide	Trivial	Separate small and big numbers
Conquer	Recurse	Recurse
Combine	Merge 2 sorted arrays	Trivial

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

29	82	75	64	38	45	94	69	25	76	15	92	37	17	85

$\text{quicksort}(A, 1, 15)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

29	82	75	64	38	45	94	69	25	76	15	92	37	17	85

$\text{quicksort}(A, 1, 15)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

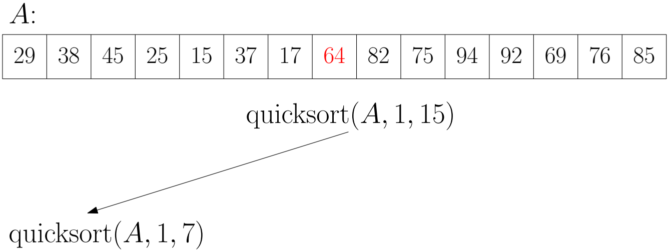
A:

29	38	45	25	15	37	17	64	82	75	94	92	69	76	85

$\text{quicksort}(A, 1, 15)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.



A:

29	38	45	25	15	37	17	64	82	75	94	92	69	76	85

quicksort$(A, 1, 15)$

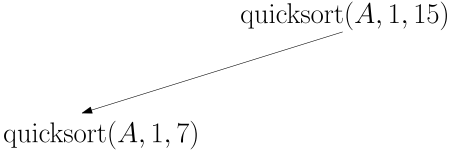
quicksort$(A, 1, 7)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

29	38	45	25	15	37	17	64	82	75	94	92	69	76	85



$\text{quicksort}(A, 1, 15)$

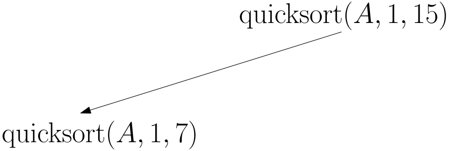
$\text{quicksort}(A, 1, 7)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

25	15	17	29	38	45	37	64	82	75	94	92	69	76	85


quicksort$(A, 1, 15)$

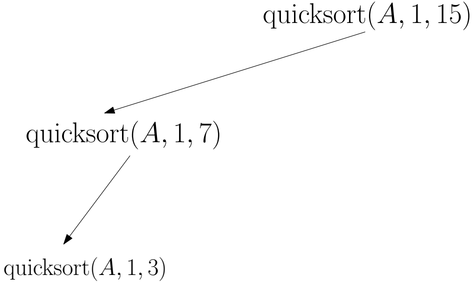
quicksort$(A, 1, 7)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

25	15	17	29	38	45	37	64	82	75	94	92	69	76	85

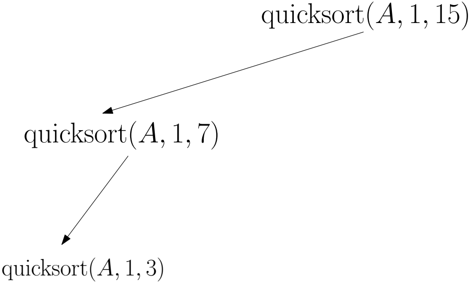


Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

15	17	25	29	38	45	37	64	82	75	94	92	69	76	85



quicksort$(A, 1, 15)$

quicksort$(A, 1, 7)$

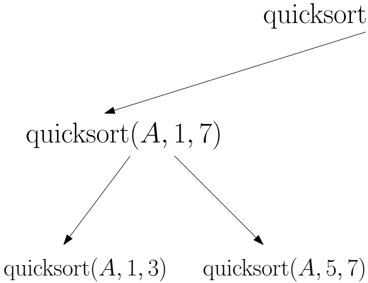
quicksort$(A, 1, 3)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

15	17	25	29	38	45	37	64	82	75	94	92	69	76	85



quicksort$(A, 1, 15)$

quicksort$(A, 1, 7)$

quicksort$(A, 1, 3)$

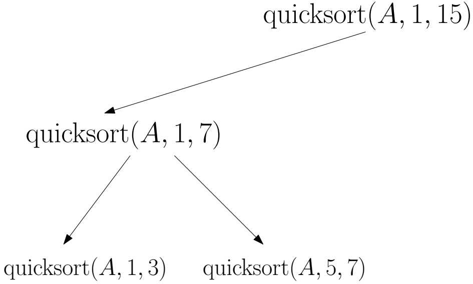
quicksort$(A, 5, 7)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

15	17	25	29	37	38	45	64	82	75	94	92	69	76	85



quicksort$(A, 1, 15)$

quicksort$(A, 1, 7)$

quicksort$(A, 1, 3)$

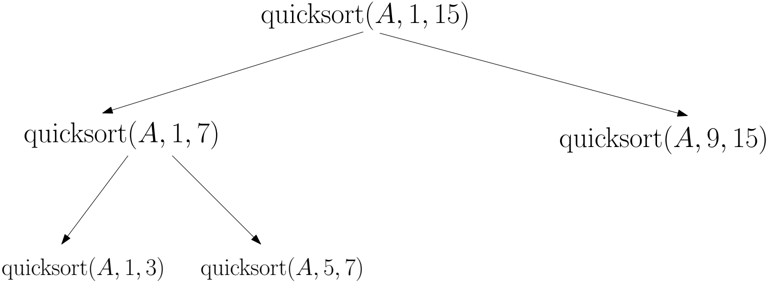
quicksort$(A, 5, 7)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

15	17	25	29	37	38	45	64	82	75	94	92	69	76	85



quicksort$(A, 1, 15)$

quicksort$(A, 1, 7)$

quicksort$(A, 9, 15)$

quicksort$(A, 1, 3)$

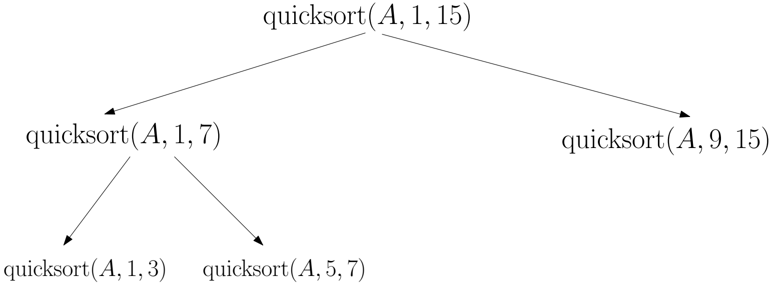
quicksort$(A, 5, 7)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

15	17	25	29	37	38	45	64	82	75	94	92	69	76	85

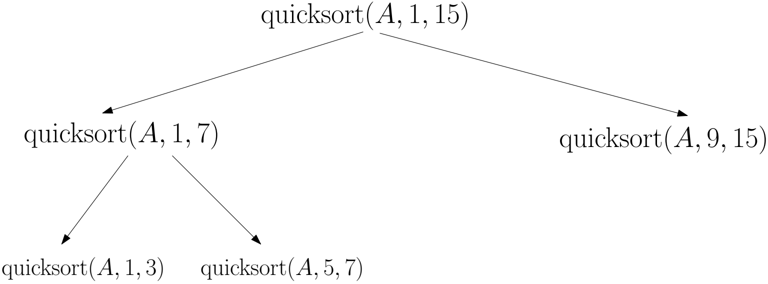


Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

15	17	25	29	37	38	45	64	75	69	76	82	92	94	85



quicksort$(A, 1, 15)$

quicksort$(A, 1, 7)$

quicksort$(A, 9, 15)$

quicksort$(A, 1, 3)$

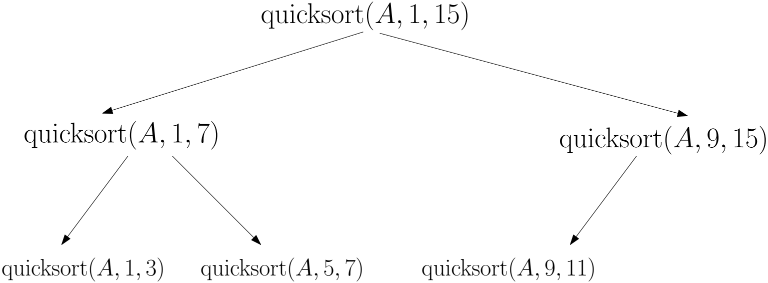
quicksort$(A, 5, 7)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

15	17	25	29	37	38	45	64	75	69	76	82	92	94	85



quicksort$(A, 1, 15)$

quicksort$(A, 1, 7)$

quicksort$(A, 9, 15)$

quicksort$(A, 1, 3)$

quicksort$(A, 5, 7)$

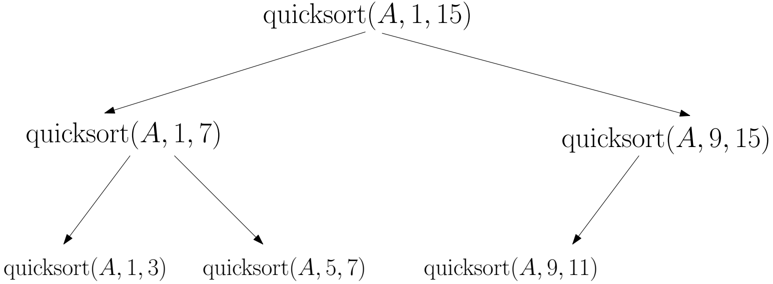
quicksort$(A, 9, 11)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

15	17	25	29	37	38	45	64	69	75	76	82	92	94	85



quicksort$(A, 1, 15)$

quicksort$(A, 1, 7)$

quicksort$(A, 9, 15)$

quicksort$(A, 1, 3)$

quicksort$(A, 5, 7)$

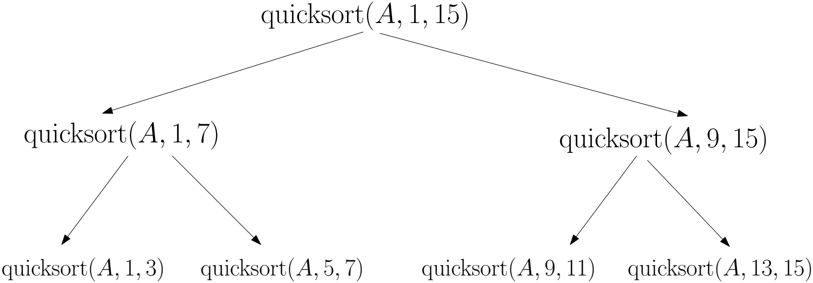
quicksort$(A, 9, 11)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

15	17	25	29	37	38	45	64	69	75	76	82	92	94	85



quicksort$(A, 1, 15)$

quicksort$(A, 1, 7)$

quicksort$(A, 9, 15)$

quicksort$(A, 1, 3)$

quicksort$(A, 5, 7)$

quicksort$(A, 9, 11)$

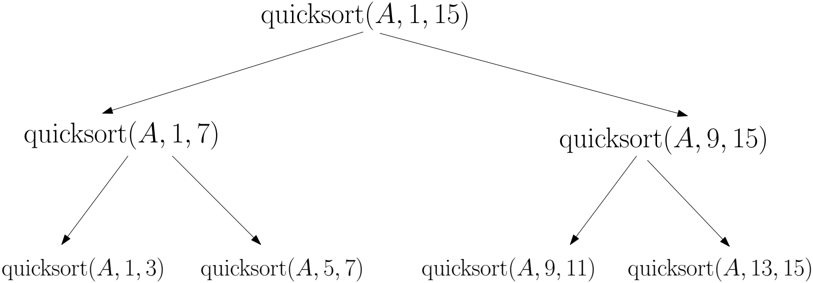
quicksort$(A, 13, 15)$

Quicksort Example

Assumption We can choose median of an array of size n in $O(n)$ time.

A:

15	17	25	29	37	38	45	64	69	75	76	82	85	92	94



quicksort$(A, 1, 15)$

quicksort$(A, 1, 7)$

quicksort$(A, 9, 15)$

quicksort$(A, 1, 3)$

quicksort$(A, 5, 7)$

quicksort$(A, 9, 11)$

quicksort$(A, 13, 15)$

Quicksort

quicksort(A, n)

```
if $n \le 1$ then return $A$
$x \leftarrow$ lower median of $A$
$A_L \leftarrow$ array of elements in $A$ that are less than $x$        \\ Divide
$A_R \leftarrow$ array of elements in $A$ that are greater than $x$     \\ Divide
$B_L \leftarrow$ quicksort$(A_L,$ length of $A_L)$                      \\ Conquer
$B_R \leftarrow$ quicksort$(A_R,$ length of $A_R)$                      \\ Conquer
$t \leftarrow$ number of times $x$ appear $A$
return concatenation of $B_L$, $t$ copies of $x$, and $B_R$
```

Quicksort

quicksort(A, n)

```
if $n \leq 1$ then return $A$
$x \leftarrow$ lower median of $A$
$A_L \leftarrow$ array of elements in $A$ that are less than $x$        \\ Divide
$A_R \leftarrow$ array of elements in $A$ that are greater than $x$     \\ Divide
$B_L \leftarrow$ quicksort$(A_L,$ length of $A_L)$                      \\ Conquer
$B_R \leftarrow$ quicksort$(A_R,$ length of $A_R)$                      \\ Conquer
$t \leftarrow$ number of times $x$ appear $A$
return concatenation of $B_L$, $t$ copies of $x$, and $B_R$
```

- Recurrence $T(n) \leq 2T(n/2) + O(n)$

Quicksort

quicksort(A, n)

```
if $n \le 1$ **then return** $A$
$x \leftarrow$ lower median of $A$
$A_L \leftarrow$ array of elements in $A$ that are less than $x$        \\ Divide
$A_R \leftarrow$ array of elements in $A$ that are greater than $x$     \\ Divide
$B_L \leftarrow$ quicksort$(A_L,$ length of $A_L)$                      \\ Conquer
$B_R \leftarrow$ quicksort$(A_R,$ length of $A_R)$                      \\ Conquer
$t \leftarrow$ number of times $x$ appear $A$
**return** concatenation of $B_L$, $t$ copies of $x$, and $B_R$
```

- Recurrence $T(n) \le 2T(n/2) + O(n)$
- Running time $= O(n \log n)$

Assumption We can choose median of an array of size n in $O(n)$ time.

Q: How to remove this assumption?

Assumption We can choose median of an array of size n in $O(n)$ time.

Q: How to remove this assumption?

A:

1. There is an algorithm to find median in $O(n)$ time, using divide-and-conquer (we shall not talk about it; it is complicated and not practical)

Assumption We can choose median of an array of size n in $O(n)$ time.

Q: How to remove this assumption?

A:

1. There is an algorithm to find median in $O(n)$ time, using divide-and-conquer (we shall not talk about it; it is complicated and not practical)
2. Choose a pivot randomly and pretend it is the median (it is practical)

Quicksort Using A Random Pivot

quicksort(A, n)

```
if $n \leq 1$ then return $A$
$x \leftarrow$ a random element of $A$ ($x$ is called a pivot)
$A_L \leftarrow$ array of elements in $A$ that are less than $x$        \\ Divide
$A_R \leftarrow$ array of elements in $A$ that are greater than $x$     \\ Divide
$B_L \leftarrow$ quicksort$(A_L,$ length of $A_L)$                       \\ Conquer
$B_R \leftarrow$ quicksort$(A_R,$ length of $A_R)$                       \\ Conquer
$t \leftarrow$ number of times $x$ appear $A$
return concatenation of $B_L$, $t$ copies of $x$, and $B_R$
```

Randomized Algorithm Model

Assumption There is a procedure to produce a random real number in $[0, 1]$.

Q: Can computers really produce random numbers?

Randomized Algorithm Model

Assumption There is a procedure to produce a random real number in $[0, 1]$.

Q: Can computers really produce random numbers?

A: No! The execution of a computer programs is deterministic!

Randomized Algorithm Model

Assumption There is a procedure to produce a random real number in $[0, 1]$.

Q: Can computers really produce random numbers?

A: No! The execution of a computer programs is deterministic!

- In practice: use pseudo-random-generator, a deterministic algorithm returning numbers that "look like" random

Randomized Algorithm Model

Assumption There is a procedure to produce a random real number in $[0, 1]$.

Q: Can computers really produce random numbers?

A: No! The execution of a computer programs is deterministic!

- In practice: use pseudo-random-generator, a deterministic algorithm returning numbers that “look like” random
- In theory: assume they can.

Quicksort Using A Random Pivot

quicksort(A, n)

```
if $n \leq 1$ then return $A$
$x \leftarrow$ a random element of $A$ ($x$ is called a pivot)
$A_L \leftarrow$ array of elements in $A$ that are less than $x$          \\ Divide
$A_R \leftarrow$ array of elements in $A$ that are greater than $x$       \\ Divide
$B_L \leftarrow$ quicksort$(A_L,$ length of $A_L)$                       \\ Conquer
$B_R \leftarrow$ quicksort$(A_R,$ length of $A_R)$                       \\ Conquer
$t \leftarrow$ number of times $x$ appear $A$
return concatenation of $B_L$, $t$ copies of $x$, and $B_R$
```

Lemma The expected running time of the algorithm is $O(n \log n)$.

Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

- In-Place Sorting Algorithm: an algorithm that only uses “small” extra space.

Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.

29	82	75	64	38	45	94	69	25	76	15	92	37	17	85

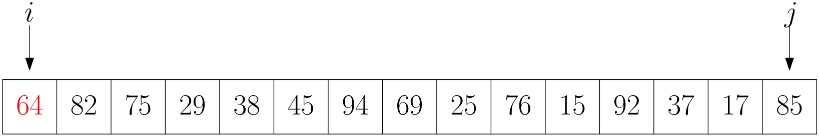
Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.

64	82	75	29	38	45	94	69	25	76	15	92	37	17	85

Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.

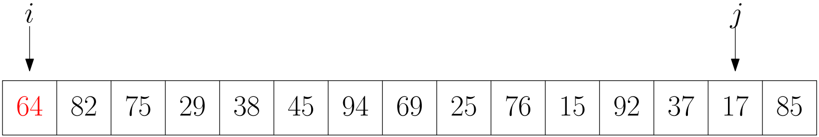


i j

64	82	75	29	38	45	94	69	25	76	15	92	37	17	85

Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

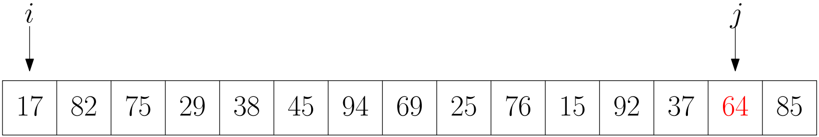
- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.



64	82	75	29	38	45	94	69	25	76	15	92	37	17	85

Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

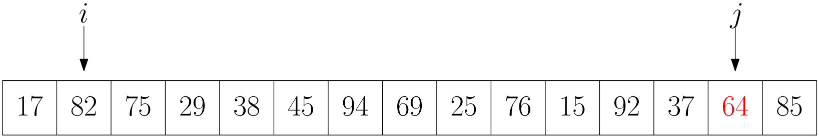
- In-Place Sorting Algorithm: an algorithm that only uses “small” extra space.



17	82	75	29	38	45	94	69	25	76	15	92	37	64	85

Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

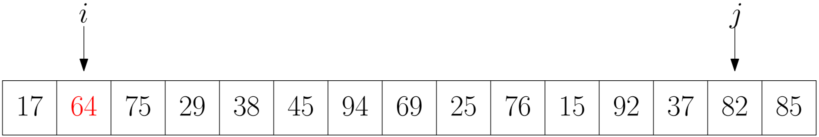
- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.



17	82	75	29	38	45	94	69	25	76	15	92	37	64	85

Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

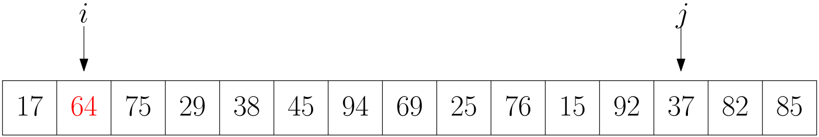
- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.



17	64	75	29	38	45	94	69	25	76	15	92	37	82	85

Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

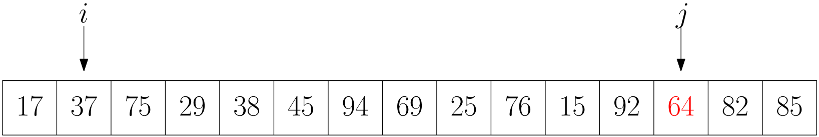
- In-Place Sorting Algorithm: an algorithm that only uses “small” extra space.



17	64	75	29	38	45	94	69	25	76	15	92	37	82	85

Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

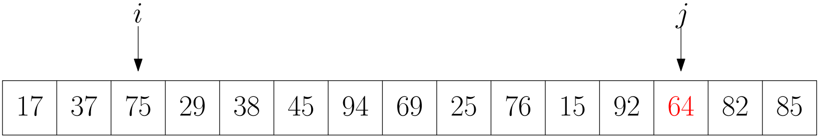
- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.



17	37	75	29	38	45	94	69	25	76	15	92	64	82	85

Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

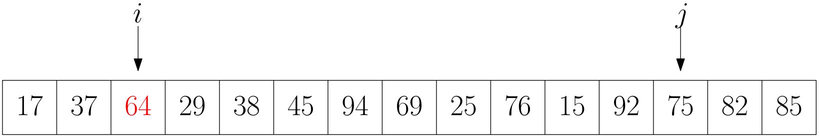
- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.



17	37	75	29	38	45	94	69	25	76	15	92	64	82	85

Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

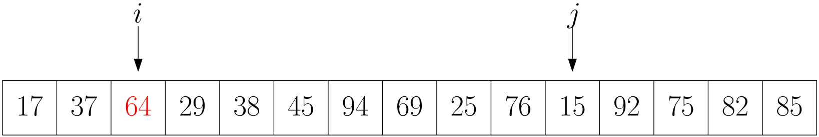
- In-Place Sorting Algorithm: an algorithm that only uses “small” extra space.



17	37	64	29	38	45	94	69	25	76	15	92	75	82	85

Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

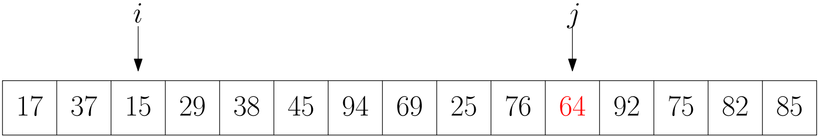
- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.



17	37	64	29	38	45	94	69	25	76	15	92	75	82	85

Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

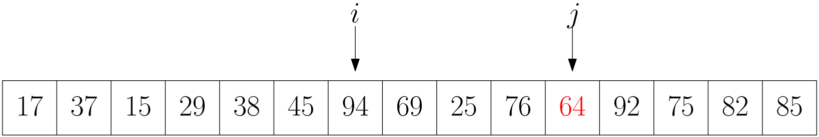
- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.



17	37	15	29	38	45	94	69	25	76	64	92	75	82	85

Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

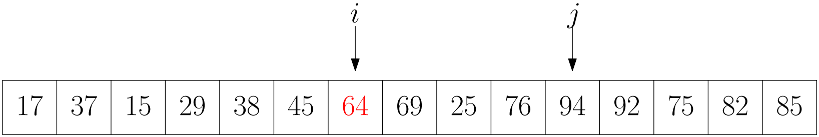
- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.



17	37	15	29	38	45	94	69	25	76	64	92	75	82	85

Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

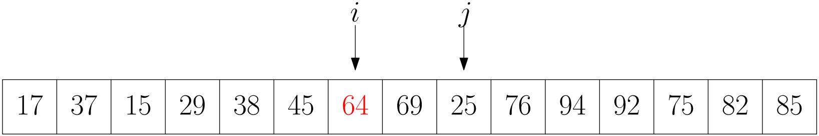
- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.



17	37	15	29	38	45	64	69	25	76	94	92	75	82	85

Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

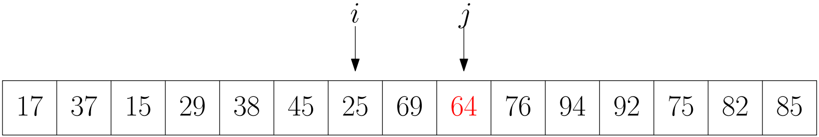
- In-Place Sorting Algorithm: an algorithm that only uses “small” extra space.



17	37	15	29	38	45	64	69	25	76	94	92	75	82	85

Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

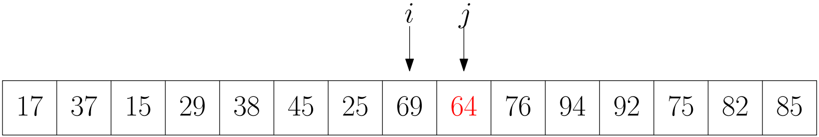
- In-Place Sorting Algorithm: an algorithm that only uses “small” extra space.



17	37	15	29	38	45	25	69	64	76	94	92	75	82	85

Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

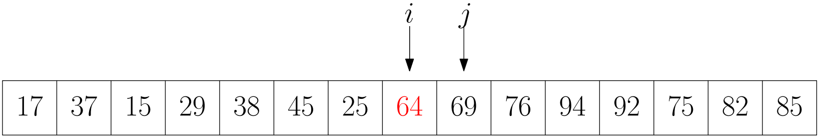
- In-Place Sorting Algorithm: an algorithm that only uses “small” extra space.



17	37	15	29	38	45	25	69	64	76	94	92	75	82	85

Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

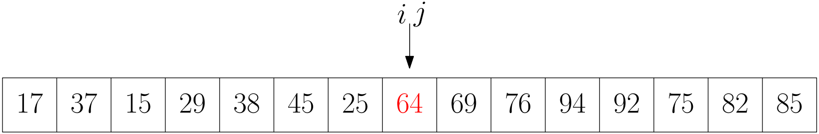
- In-Place Sorting Algorithm: an algorithm that only uses “small” extra space.



17	37	15	29	38	45	25	64	69	76	94	92	75	82	85

Quicksort Can Be Implemented as an "In-Place" Sorting Algorithm

- In-Place Sorting Algorithm: an algorithm that only uses "small" extra space.

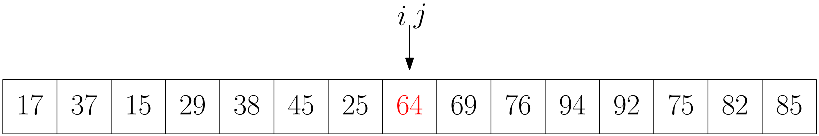


i j

17	37	15	29	38	45	25	64	69	76	94	92	75	82	85

Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

- In-Place Sorting Algorithm: an algorithm that only uses “small” extra space.



17	37	15	29	38	45	25	64	69	76	94	92	75	82	85

- To partition the array into two parts, we only need $O(1)$ extra space.

partition(A, ℓ, r)

```
$p \leftarrow$ random integer between $\ell$ and $r$, swap $A[p]$ and $A[\ell]$
$i \leftarrow \ell, j \leftarrow r$
while true do
    while $i < j$ and $A[i] < A[j]$ do $j \leftarrow j - 1$
    if $i = j$ then break
    swap $A[i]$ and $A[j]$; $i \leftarrow i + 1$
    while $i < j$ and $A[i] < A[j]$ do $i \leftarrow i + 1$
    if $i = j$ then break
    swap $A[i]$ and $A[j]$; $j \leftarrow j - 1$
return $i$
```

In-Place Implementation of Quick-Sort

quicksort(A, ℓ, r)

```
if $\ell \geq r$ then return
$m \leftarrow$ patition$(A, \ell, r)$
quicksort$(A, \ell, m - 1)$
quicksort$(A, m + 1, r)$
```

- To sort an array A of size n, call quicksort$(A, 1, n)$.

Note: We pass the array A by reference, instead of by copying.

Merge-Sort is Not In-Place

- To merge two arrays, we need a third array with size equaling the total size of two arrays

Merge-Sort is Not In-Place

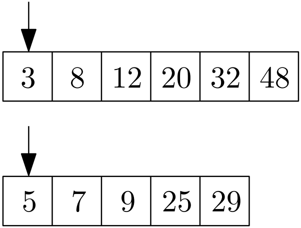
- To merge two arrays, we need a third array with size equaling the total size of two arrays

3	8	12	20	32	48

5	7	9	25	29

Merge-Sort is Not In-Place

- To merge two arrays, we need a third array with size equaling the total size of two arrays

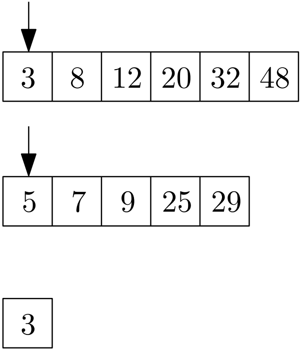


3	8	12	20	32	48

5	7	9	25	29

Merge-Sort is Not In-Place

- To merge two arrays, we need a third array with size equaling the total size of two arrays



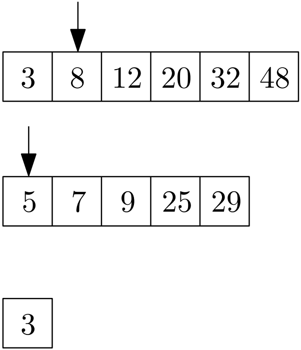
3	8	12	20	32	48

5	7	9	25	29

3

Merge-Sort is Not In-Place

- To merge two arrays, we need a third array with size equaling the total size of two arrays



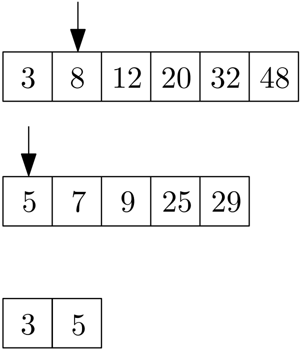
3	8	12	20	32	48

5	7	9	25	29

3

Merge-Sort is Not In-Place

- To merge two arrays, we need a third array with size equaling the total size of two arrays



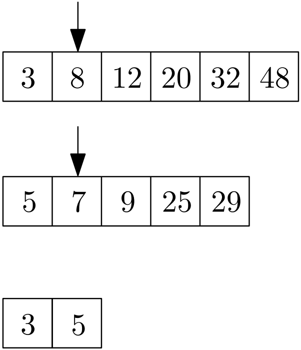
3	8	12	20	32	48

5	7	9	25	29

3	5

Merge-Sort is Not In-Place

- To merge two arrays, we need a third array with size equaling the total size of two arrays



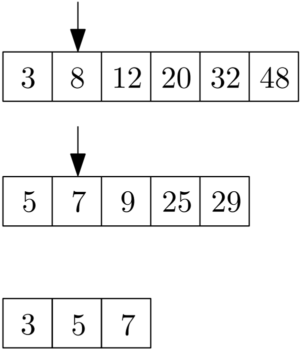
3 8 12 20 32 48

5 7 9 25 29

3 5

Merge-Sort is Not In-Place

- To merge two arrays, we need a third array with size equaling the total size of two arrays



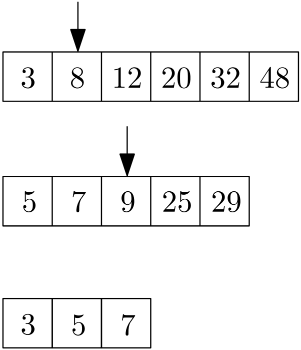
3	8	12	20	32	48

5	7	9	25	29

3	5	7

Merge-Sort is Not In-Place

- To merge two arrays, we need a third array with size equaling the total size of two arrays



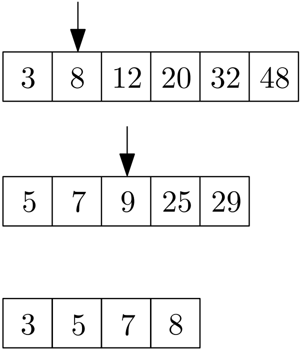
3	8	12	20	32	48

5	7	9	25	29

3	5	7

Merge-Sort is Not In-Place

- To merge two arrays, we need a third array with size equaling the total size of two arrays



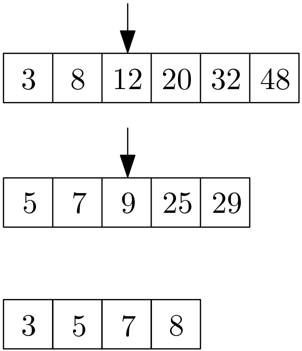
3	8	12	20	32	48

5	7	9	25	29

3	5	7	8

Merge-Sort is Not In-Place

- To merge two arrays, we need a third array with size equaling the total size of two arrays



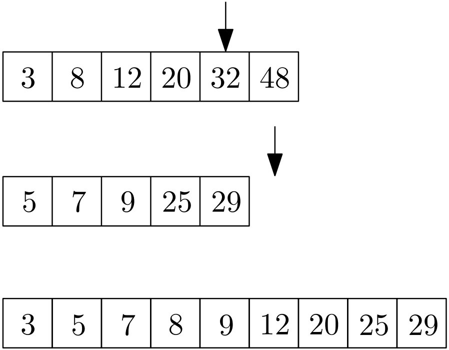
3	8	12	20	32	48

5	7	9	25	29

3	5	7	8

Merge-Sort is Not In-Place

- To merge two arrays, we need a third array with size equaling the total size of two arrays



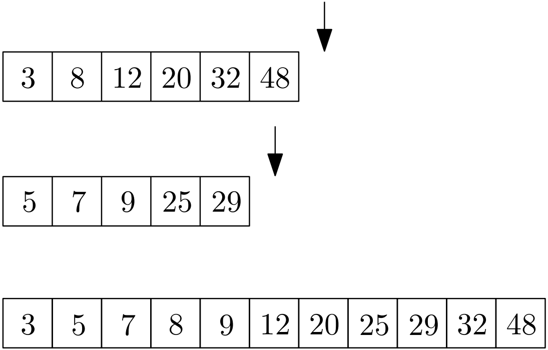
3	8	12	20	32	48

5	7	9	25	29

3	5	7	8	9	12	20	25	29

Merge-Sort is Not In-Place

- To merge two arrays, we need a third array with size equaling the total size of two arrays



3	8	12	20	32	48

5	7	9	25	29

3	5	7	8	9	12	20	25	29	32	48

Outline

1. Divide-and-Conquer
2. Counting Inversions
3. Solving Recurrences
4. Quicksort and Selection
   - Quicksort
   - Lower Bound for Comparison-Based Sorting Algorithms
   - Selection Problem
5. Polynomial Multiplication
6. Strassen's Algorithm for Matrix Multiplication
7. FFT(Fast Fourier Transform): Polynomial Multiplication in $O(n \log n)$ Time
8. Finding Closest Pair of Points in 2D Euclidean Space
9. Computing n-th Fibonacci Number

Comparison-Based Sorting Algorithms

Q: Can we do better than $O(n \log n)$ for sorting?

Comparison-Based Sorting Algorithms

Q: Can we do better than $O(n \log n)$ for sorting?

A: No, for comparison-based sorting algorithms.

Comparison-Based Sorting Algorithms

Q: Can we do better than $O(n \log n)$ for sorting?

A: No, for comparison-based sorting algorithms.

Comparison-Based Sorting Algorithms

- To sort, we are only allowed to compare two elements
- We can not use "internal structures" of the elements

Lemma The (worst-case) running time of any comparison-based sorting algorithm is $\Omega(n \log n)$.

Lemma The (worst-case) running time of any comparison-based sorting algorithm is $\Omega(n \log n)$.

- Bob has one number x in his hand, $x \in \{1, 2, 3, \cdots, N\}$.

Lemma The (worst-case) running time of any comparison-based sorting algorithm is $\Omega(n \log n)$.

- Bob has one number x in his hand, $x \in \{1, 2, 3, \cdots, N\}$.
- You can ask Bob "yes/no" questions about x.

Lemma The (worst-case) running time of any comparison-based sorting algorithm is $\Omega(n \log n)$.

- Bob has one number x in his hand, $x \in \{1, 2, 3, \cdots, N\}$.
- You can ask Bob "yes/no" questions about x.

Q: How many questions do you need to ask Bob in order to know x?

Lemma The (worst-case) running time of any comparison-based sorting algorithm is $\Omega(n \log n)$.

- Bob has one number x in his hand, $x \in \{1, 2, 3, \cdots, N\}$.
- You can ask Bob "yes/no" questions about x.

Q: How many questions do you need to ask Bob in order to know x?

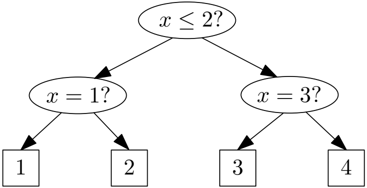
A: $\lceil \log_2 N \rceil$.

Lemma The (worst-case) running time of any comparison-based sorting algorithm is $\Omega(n \log n)$.

- Bob has one number x in his hand, $x \in \{1, 2, 3, \cdots, N\}$.
- You can ask Bob "yes/no" questions about x.

Q: How many questions do you need to ask Bob in order to know x?

A: $\lceil \log_2 N \rceil$.



Comparison-Based Sorting Algorithms

Q: Can we do better than $O(n \log n)$ for sorting?

A: No, for comparison-based sorting algorithms.

- Bob has a permutation π over $\{1, 2, 3, \cdots, n\}$ in his hand.
- You can ask Bob "yes/no" questions about π.

Comparison-Based Sorting Algorithms

Q: Can we do better than $O(n \log n)$ for sorting?

A: No, for comparison-based sorting algorithms.

- Bob has a permutation π over $\{1, 2, 3, \cdots, n\}$ in his hand.
- You can ask Bob "yes/no" questions about π.

Q: How many questions do you need to ask in order to get the permutation π?

Comparison-Based Sorting Algorithms

Q: Can we do better than $O(n \log n)$ for sorting?

A: No, for comparison-based sorting algorithms.

- Bob has a permutation π over $\{1, 2, 3, \cdots, n\}$ in his hand.
- You can ask Bob "yes/no" questions about π.

Q: How many questions do you need to ask in order to get the permutation π?

A: $\log_2 n! = \Theta(n \log n)$

Comparison-Based Sorting Algorithms

Q: Can we do better than $O(n \log n)$ for sorting?

A: No, for comparison-based sorting algorithms.

- Bob has a permutation π over $\{1, 2, 3, \cdots, n\}$ in his hand.
- You can ask Bob questions of the form "does i appear before j in π?"

Comparison-Based Sorting Algorithms

Q: Can we do better than $O(n \log n)$ for sorting?

A: No, for comparison-based sorting algorithms.

- Bob has a permutation π over $\{1, 2, 3, \cdots, n\}$ in his hand.
- You can ask Bob questions of the form "does i appear before j in π?"

Q: How many questions do you need to ask in order to get the permutation π?

Comparison-Based Sorting Algorithms

Q: Can we do better than $O(n \log n)$ for sorting?

A: No, for comparison-based sorting algorithms.

- Bob has a permutation π over $\{1, 2, 3, \cdots, n\}$ in his hand.
- You can ask Bob questions of the form "does i appear before j in π?"

Q: How many questions do you need to ask in order to get the permutation π?

A: At least $\log_2 n! = \Theta(n \log n)$

Outline

1. Divide-and-Conquer
2. Counting Inversions
3. Solving Recurrences
4. Quicksort and Selection
    - Quicksort
    - Lower Bound for Comparison-Based Sorting Algorithms
    - Selection Problem
5. Polynomial Multiplication
6. Strassen's Algorithm for Matrix Multiplication
7. FFT(Fast Fourier Transform): Polynomial Multiplication in $O(n \log n)$ Time
8. Finding Closest Pair of Points in 2D Euclidean Space
9. Computing n-th Fibonacci Number

Selection Problem

Input: a set A of n numbers, and $1 \le i \le n$

Output: the i-th smallest number in A

Selection Problem

Input: a set A of n numbers, and $1 \le i \le n$

Output: the i-th smallest number in A

- Sorting solves the problem in time $O(n \log n)$.

Selection Problem

Input: a set A of n numbers, and $1 \le i \le n$

Output: the i-th smallest number in A

- Sorting solves the problem in time $O(n \log n)$.
- Our goal: $O(n)$ running time

Recall: Quicksort with Median Finder

```
quicksort($A, n$)
if $n \leq 1$ then return $A$
$x \leftarrow$ lower median of $A$
$A_L \leftarrow$ elements in $A$ that are less than $x$        ▷ Divide
$A_R \leftarrow$ elements in $A$ that are greater than $x$     ▷ Divide
$B_L \leftarrow$ quicksort($A_L, A_L$.size)                     ▷ Conquer
$B_R \leftarrow$ quicksort($A_R, A_R$.size)                     ▷ Conquer
$t \leftarrow$ number of times $x$ appear $A$
return the array obtained by concatenating $B_L$, the array containing $t$ copies of $x$, and $B_R$
```

Selection Algorithm with Median Finder

selection(A, n, i)

```
if $n = 1$ then return $A$
$x \leftarrow$ lower median of $A$
$A_L \leftarrow$ elements in $A$ that are less than $x$                ▷ Divide
$A_R \leftarrow$ elements in $A$ that are greater than $x$             ▷ Divide
if $i \leq A_L$.size then
    return selection$(A_L, A_L.\text{size}, i)$                         ▷ Conquer
else if $i > n - A_R$.size then
    return selection$(A_R, A_R.\text{size}, i - (n - A_R.\text{size}))$  ▷ Conquer
else
    return $x$
```

Selection Algorithm with Median Finder

selection(A, n, i)

```
if $n = 1$ then return $A$
$x \leftarrow$ lower median of $A$
$A_L \leftarrow$ elements in $A$ that are less than $x$          ▷ Divide
$A_R \leftarrow$ elements in $A$ that are greater than $x$       ▷ Divide
if $i \leq A_L$.size then
    return selection$(A_L, A_L$.size$, i)$                        ▷ Conquer
else if $i > n - A_R$.size then
    return selection$(A_R, A_R$.size$, i - (n - A_R$.size$))$     ▷ Conquer
else
    return $x$
```

- Recurrence for selection: $T(n) = T(n/2) + O(n)$

Selection Algorithm with Median Finder

```
selection$(A, n, i)$
if $n = 1$ then return $A$
$x \leftarrow$ lower median of $A$
$A_L \leftarrow$ elements in $A$ that are less than $x$                ▷ Divide
$A_R \leftarrow$ elements in $A$ that are greater than $x$             ▷ Divide
if $i \leq A_L$.size then
    return selection$(A_L, A_L.\text{size}, i)$                        ▷ Conquer
else if $i > n - A_R$.size then
    return selection$(A_R, A_R.\text{size}, i - (n - A_R.\text{size}))$  ▷ Conquer
else
    return $x$
```

- Recurrence for selection: $T(n) = T(n/2) + O(n)$
- Solving recurrence: $T(n) = O(n)$

Randomized Selection Algorithm

```
selection($A, n, i$)
if $n = 1$ thenreturn $A$
$x \leftarrow$ random element of $A$ (called pivot)
$A_L \leftarrow$ elements in $A$ that are less than $x$          ▷ Divide
$A_R \leftarrow$ elements in $A$ that are greater than $x$       ▷ Divide
if $i \leq A_L$.size then
    return selection($A_L, A_L$.size$, i$)                       ▷ Conquer
else if $i > n - A_R$.size then
    return selection($A_R, A_R$.size$, i - (n - A_R$.size$)$)    ▷ Conquer
else
    return $x$
```

Randomized Selection Algorithm

selection(A, n, i)

```
if $n = 1$ thenreturn $A$
$x \leftarrow$ random element of $A$ (called pivot)
$A_L \leftarrow$ elements in $A$ that are less than $x$            ▷ Divide
$A_R \leftarrow$ elements in $A$ that are greater than $x$         ▷ Divide
if $i \leq A_L$.size then
    return selection$(A_L, A_L.\text{size}, i)$                    ▷ Conquer
else if $i > n - A_R$.size then
    return selection$(A_R, A_R.\text{size}, i - (n - A_R.\text{size}))$    ▷ Conquer
else
    return $x$
```

- expected running time $= O(n)$

Outline

1. Divide-and-Conquer
2. Counting Inversions
3. Solving Recurrences
4. Quicksort and Selection
   - Quicksort
   - Lower Bound for Comparison-Based Sorting Algorithms
   - Selection Problem
5. **Polynomial Multiplication**
6. Strassen's Algorithm for Matrix Multiplication
7. FFT(Fast Fourier Transform): Polynomial Multiplication in $O(n \log n)$ Time
8. Finding Closest Pair of Points in 2D Euclidean Space
9. Computing n-th Fibonacci Number

Polynomial Multiplication

Input: two polynomials of degree $n-1$

Output: product of two polynomials

Polynomial Multiplication

Input: two polynomials of degree $n-1$

Output: product of two polynomials

Example:

$$(3x^3 + 2x^2 - 5x + 4) \times (2x^3 - 3x^2 + 6x - 5)$$

Polynomial Multiplication

Input: two polynomials of degree $n-1$

Output: product of two polynomials

Example:

$$
\begin{aligned}
&(3x^3+2x^2-5x+4)\times(2x^3-3x^2+6x-5)\\
={}&6x^6-9x^5+18x^4-15x^3\\
&+4x^5-6x^4+12x^3-10x^2\\
&-10x^4+15x^3-30x^2+25x\\
&+8x^3-12x^2+24x-20\\
={}&6x^6-5x^5+2x^4+20x^3-52x^2+49x-20
\end{aligned}
$$

Polynomial Multiplication

Input: two polynomials of degree $n-1$

Output: product of two polynomials

Example:

$$\begin{aligned}
&(3x^3+2x^2-5x+4)\times(2x^3-3x^2+6x-5)\\
=\ &6x^6-9x^5+18x^4-15x^3\\
&+4x^5-6x^4+12x^3-10x^2\\
&-10x^4+15x^3-30x^2+25x\\
&+8x^3-12x^2+24x-20\\
=\ &6x^6-5x^5+2x^4+20x^3-52x^2+49x-20
\end{aligned}$$

- **Input**: $(4,-5,2,3),(-5,6,-3,2)$
- **Output**: $(-20,49,-52,20,2,-5,6)$

Discrete Convolution on Finite Domain

- $f : \{0, 1, \cdots, n-1\} \to \mathbb{R}, g : \{0, 1, \cdots, m-1\} \to \mathbb{R}$
- the convolution of f and g, denoted as $h := f \times g$, is defined as

$$h(k) := \sum_{i,j:i+j=k} f(i)g(j) \qquad \forall k \in \{0, 1, 2, \cdots, m+n-2\}$$

Discrete Convolution on Finite Domain

- $f : \{0, 1, \cdots, n-1\} \to \mathbb{R}, g : \{0, 1, \cdots, m-1\} \to \mathbb{R}$
- the convolution of f and g, denoted as $h := f \times g$, is defined as

$$h(k) := \sum_{i,j:i+j=k} f(i)g(j) \qquad \forall k \in \{0, 1, 2, \cdots, m+n-2\}$$

	0	1	2	3	4	5	6
f	4	-5	2	3			
g	-5	6	-3	2			
$f \times g$	-20	49	-52	20	2	-5	6

Discrete Convolution on Finite Domain

- $f : \{0, 1, \cdots, n-1\} \to \mathbb{R}, g : \{0, 1, \cdots, m-1\} \to \mathbb{R}$
- the convolution of f and g, denoted as $h := f \times g$, is defined as

$$h(k) := \sum_{i,j : i+j=k} f(i)g(j) \qquad \forall k \in \{0, 1, 2, \cdots, m+n-2\}$$

	0	1	2	3	4	5	6
f	4	-5	2	3			
g	-5	6	-3	2			
$f \times g$	-20	49	-52	20	2	-5	6

Applications of Convolutions

- Polynomial and integer multiplication
- Signal and Image Processing
- Probability theory: Sum of two distributions
- Convolutional neural network

- Polynomial multiplication ⇔ Convolution
- We shall focus on multiplication.

Big Integer Multiplication Using Polynomial Multiplication

- Polynomial multiplication ⇔ Convolution
- We shall focus on multiplication.

Big Integer Multiplication Using Polynomial Multiplication

- $16103416169 \times 424317167$

- Polynomial multiplication ⇔ Convolution
- We shall focus on multiplication.

Big Integer Multiplication Using Polynomial Multiplication

- $16103416169 \times 424317167$
- $(16x^3 + 103x^2 + 416x + 169) \times (424x^2 + 317x + 167)$

- Polynomial multiplication ⇔ Convolution
- We shall focus on multiplication.

Big Integer Multiplication Using Polynomial Multiplication

- $16103416169 \times 424317167$
- $(16x^3 + 103x^2 + 416x + 169) \times (424x^2 + 317x + 167)$
- $6784x^5 + 48744x^4 + 211707x^3 + 220729x^2 + 123045x + 28223$

- Polynomial multiplication ⇔ Convolution
- We shall focus on multiplication.

Big Integer Multiplication Using Polynomial Multiplication

- $16103416169 \times 424317167$
- $(16x^3 + 103x^2 + 416x + 169) \times (424x^2 + 317x + 167)$
- $6784x^5 + 48744x^4 + 211707x^3 + 220729x^2 + 123045x + 28223$
- $6784, 48744, 211707, 220729, 123045, 28223$

- Polynomial multiplication ⇔ Convolution
- We shall focus on multiplication.

Big Integer Multiplication Using Polynomial Multiplication

- $16103416169 \times 424317167$
- $(16x^3 + 103x^2 + 416x + 169) \times (424x^2 + 317x + 167)$
- $6784x^5 + 48744x^4 + 211707x^3 + 220729x^2 + 123045x + 28223$
- $6784, 48744, 211707, 220729, 123073, 223$

- Polynomial multiplication ⇔ Convolution
- We shall focus on multiplication.

Big Integer Multiplication Using Polynomial Multiplication

- $16103416169 \times 424317167$
- $(16x^3 + 103x^2 + 416x + 169) \times (424x^2 + 317x + 167)$
- $6784x^5 + 48744x^4 + 211707x^3 + 220729x^2 + 123045x + 28223$
- $6784, 48744, 211707, 220852, 073, 223$

- Polynomial multiplication ⇔ Convolution
- We shall focus on multiplication.

Big Integer Multiplication Using Polynomial Multiplication

- $16103416169 \times 424317167$
- $(16x^3 + 103x^2 + 416x + 169) \times (424x^2 + 317x + 167)$
- $6784x^5 + 48744x^4 + 211707x^3 + 220729x^2 + 123045x + 28223$
- $6784, 48744, 211927, 852, 073, 223$

- Polynomial multiplication $\Leftrightarrow$ Convolution
- We shall focus on multiplication.

Big Integer Multiplication Using Polynomial Multiplication

- $16103416169 \times 424317167$
- $(16x^3 + 103x^2 + 416x + 169) \times (424x^2 + 317x + 167)$
- $6784x^5 + 48744x^4 + 211707x^3 + 220729x^2 + 123045x + 28223$
- $6784, 48955, 927, 852, 073, 223$

- Polynomial multiplication ⇔ Convolution
- We shall focus on multiplication.

Big Integer Multiplication Using Polynomial Multiplication

- $16103416169 \times 424317167$
- $(16x^3 + 103x^2 + 416x + 169) \times (424x^2 + 317x + 167)$
- $6784x^5 + 48744x^4 + 211707x^3 + 220729x^2 + 123045x + 28223$
- $6832, 955, 927, 852, 073, 223$

- Polynomial multiplication ⇔ Convolution
- We shall focus on multiplication.

Big Integer Multiplication Using Polynomial Multiplication

- $16103416169 \times 424317167$
- $(16x^3 + 103x^2 + 416x + 169) \times (424x^2 + 317x + 167)$
- $6784x^5 + 48744x^4 + 211707x^3 + 220729x^2 + 123045x + 28223$
- $6832, 955, 927, 852, 073, 223$
- 6832955927852073223

Naïve Algorithm

polynomial-multiplication(A, B, n)

```
let $C[k] \leftarrow 0$ for every $k = 0, 1, 2, \cdots, 2n-2$
for $i \leftarrow 0$ to $n-1$ do
    for $j \leftarrow 0$ to $n-1$ do
        $C[i+j] \leftarrow C[i+j] + A[i] \times B[j]$
return $C$
```

Naïve Algorithm

```
polynomial-multiplication($A, B, n$)
let $C[k] \leftarrow 0$ for every $k = 0, 1, 2, \cdots, 2n-2$
for $i \leftarrow 0$ to $n-1$ do
    for $j \leftarrow 0$ to $n-1$ do
        $C[i+j] \leftarrow C[i+j] + A[i] \times B[j]$
return $C$
```

Running time: $O(n^2)$

Divide-and-Conquer for Polynomial Multiplication

$$p(x) = 3x^3 + 2x^2 - 5x + 4 = (3x+2)x^2 + (-5x+4)$$
$$q(x) = 2x^3 - 3x^2 + 6x - 5 = (2x-3)x^2 + (6x-5)$$

Divide-and-Conquer for Polynomial Multiplication

$$p(x) = 3x^3 + 2x^2 - 5x + 4 = (3x + 2)x^2 + (-5x + 4)$$
$$q(x) = 2x^3 - 3x^2 + 6x - 5 = (2x - 3)x^2 + (6x - 5)$$

- $p(x)$: degree of $n - 1$ (assume n is even)
- $p(x) = p_H(x)x^{n/2} + p_L(x)$,
- $p_H(x), p_L(x)$: polynomials of degree $n/2 - 1$.

Divide-and-Conquer for Polynomial Multiplication

$$p(x) = 3x^3 + 2x^2 - 5x + 4 = (3x+2)x^2 + (-5x+4)$$
$$q(x) = 2x^3 - 3x^2 + 6x - 5 = (2x-3)x^2 + (6x-5)$$

- $p(x)$: degree of $n-1$ (assume n is even)
- $p(x) = p_H(x)x^{n/2} + p_L(x)$,
- $p_H(x), p_L(x)$: polynomials of degree $n/2-1$.

$$pq = \left(p_H x^{n/2} + p_L\right)\left(q_H x^{n/2} + q_L\right)$$

Divide-and-Conquer for Polynomial Multiplication

$$p(x) = 3x^3 + 2x^2 - 5x + 4 = (3x + 2)x^2 + (-5x + 4)$$
$$q(x) = 2x^3 - 3x^2 + 6x - 5 = (2x - 3)x^2 + (6x - 5)$$

- $p(x)$: degree of $n - 1$ (assume n is even)
- $p(x) = p_H(x)x^{n/2} + p_L(x)$,
- $p_H(x), p_L(x)$: polynomials of degree $n/2 - 1$.

$$\begin{aligned} pq &= \left(p_H x^{n/2} + p_L\right)\left(q_H x^{n/2} + q_L\right) \\ &= p_H q_H x^n + \left(p_H q_L + p_L q_H\right)x^{n/2} + p_L q_L \end{aligned}$$

Divide-and-Conquer for Polynomial Multiplication

$$\begin{aligned} pq &= \left(p_H x^{n/2} + p_L\right)\left(q_H x^{n/2} + q_L\right) \\ &= p_H q_H x^n + \left(p_H q_L + p_L q_H\right) x^{n/2} + p_L q_L \end{aligned}$$

Divide-and-Conquer for Polynomial Multiplication

$$\begin{aligned} pq &= \bigl(p_H x^{n/2} + p_L\bigr)\bigl(q_H x^{n/2} + q_L\bigr) \\ &= p_H q_H x^n + \bigl(p_H q_L + p_L q_H\bigr) x^{n/2} + p_L q_L \end{aligned}$$

$$\begin{aligned} \mathsf{multiply}(p, q) = {} & \mathsf{multiply}(p_H, q_H) \times x^n \\ & + \bigl(\mathsf{multiply}(p_H, q_L) + \mathsf{multiply}(p_L, q_H)\bigr) \times x^{n/2} \\ & + \mathsf{multiply}(p_L, q_L) \end{aligned}$$

Divide-and-Conquer for Polynomial Multiplication

$$\begin{aligned} pq &= \big(p_H x^{n/2} + p_L\big)\big(q_H x^{n/2} + q_L\big) \\ &= p_H q_H x^n + \big(p_H q_L + p_L q_H\big) x^{n/2} + p_L q_L \end{aligned}$$

$$\begin{aligned} \mathsf{multiply}(p, q) = {} & \mathsf{multiply}(p_H, q_H) \times x^n \\ & + \big(\mathsf{multiply}(p_H, q_L) + \mathsf{multiply}(p_L, q_H)\big) \times x^{n/2} \\ & + \mathsf{multiply}(p_L, q_L) \end{aligned}$$

- Recurrence: $T(n) = 4T(n/2) + O(n)$

Divide-and-Conquer for Polynomial Multiplication

$$\begin{aligned} pq &= \big(p_H x^{n/2} + p_L\big)\big(q_H x^{n/2} + q_L\big) \\ &= p_H q_H x^n + \big(p_H q_L + p_L q_H\big) x^{n/2} + p_L q_L \end{aligned}$$

$$\begin{aligned} \mathsf{multiply}(p, q) = {} & \mathsf{multiply}(p_H, q_H) \times x^n \\ & + \big(\mathsf{multiply}(p_H, q_L) + \mathsf{multiply}(p_L, q_H)\big) \times x^{n/2} \\ & + \mathsf{multiply}(p_L, q_L) \end{aligned}$$

- Recurrence: $T(n) = 4T(n/2) + O(n)$
- $T(n) = O(n^2)$

Reduce Number from 4 to 3

Reduce Number from 4 to 3

$$\begin{aligned} pq &= \big(p_H x^{n/2} + p_L\big)\big(q_H x^{n/2} + q_L\big) \\ &= p_H q_H x^n + \big(p_H q_L + p_L q_H\big) x^{n/2} + p_L q_L \end{aligned}$$

Reduce Number from 4 to 3

$$
\begin{aligned}
pq &= \left(p_H x^{n/2} + p_L\right)\left(q_H x^{n/2} + q_L\right) \\
&= p_H q_H x^n + \left(p_H q_L + p_L q_H\right) x^{n/2} + p_L q_L
\end{aligned}
$$

- $p_H q_L + p_L q_H = (p_H + p_L)(q_H + q_L) - p_H q_H - p_L q_L$

Divide-and-Conquer for Polynomial Multiplication

Divide-and-Conquer for Polynomial Multiplication

$$r_H = \mathsf{multiply}(p_H, q_H)$$
$$r_L = \mathsf{multiply}(p_L, q_L)$$

Divide-and-Conquer for Polynomial Multiplication

$$r_H = \mathsf{multiply}(p_H, q_H)$$
$$r_L = \mathsf{multiply}(p_L, q_L)$$

$$\begin{aligned}\mathsf{multiply}(p, q) = {} & r_H \times x^n \\ & + \big(\mathsf{multiply}(p_H + p_L, q_H + q_L) - r_H - r_L\big) \times x^{n/2} \\ & + r_L\end{aligned}$$

Divide-and-Conquer for Polynomial Multiplication

$$r_H = \mathsf{multiply}(p_H, q_H)$$
$$r_L = \mathsf{multiply}(p_L, q_L)$$

$$\begin{aligned}\mathsf{multiply}(p,q) = {} & r_H \times x^n \\ & + \big(\mathsf{multiply}(p_H + p_L, q_H + q_L) - r_H - r_L\big) \times x^{n/2} \\ & + r_L\end{aligned}$$

- Solving Recurrence: $T(n) = 3T(n/2) + O(n)$

Divide-and-Conquer for Polynomial Multiplication

$$r_H = \mathsf{multiply}(p_H, q_H)$$
$$r_L = \mathsf{multiply}(p_L, q_L)$$

$$\begin{aligned}\mathsf{multiply}(p, q) = {} & r_H \times x^n \\ & + \big(\mathsf{multiply}(p_H + p_L, q_H + q_L) - r_H - r_L\big) \times x^{n/2} \\ & + r_L\end{aligned}$$

- Solving Recurrence: $T(n) = 3T(n/2) + O(n)$
- $T(n) = O(n^{\log_2 3}) = O(n^{1.585})$

Assumption n is a power of 2. Arrays are 0-indexed.

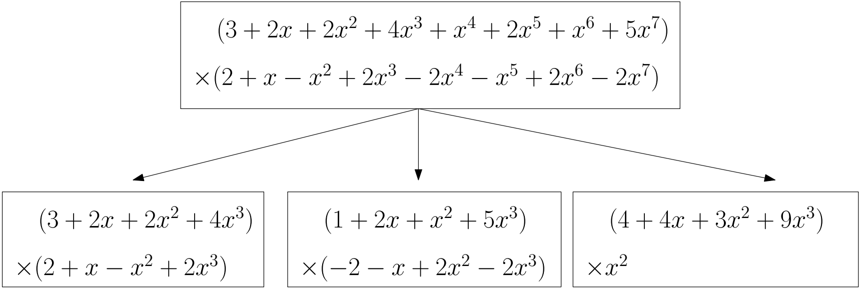
multiply(A, B, n)

```
if $n = 1$ then return $(A[0]B[0])$
$A_L \leftarrow A[0 \,..\, n/2 - 1], A_H \leftarrow A[n/2 \,..\, n - 1]$
$B_L \leftarrow B[0 \,..\, n/2 - 1], B_H \leftarrow B[n/2 \,..\, n - 1]$
$C_L \leftarrow$ multiply$(A_L, B_L, n/2)$
$C_H \leftarrow$ multiply$(A_H, B_H, n/2)$
$C_M \leftarrow$ multiply$(A_L + A_H, B_L + B_H, n/2)$
$C \leftarrow$ array of $(2n - 1)$ 0's
**for** $i \leftarrow 0$ to $n - 2$ **do**
    $C[i] \leftarrow C[i] + C_L[i]$
    $C[i + n] \leftarrow C[i + n] + C_H[i]$
    $C[i + n/2] \leftarrow C[i + n/2] + C_M[i] - C_L[i] - C_H[i]$
**return** $C$
```

Example

$$\boxed{\begin{aligned} &(3 + 2x + 2x^2 + 4x^3 + x^4 + 2x^5 + x^6 + 5x^7) \\ \times&(2 + x - x^2 + 2x^3 - 2x^4 - x^5 + 2x^6 - 2x^7) \end{aligned}}$$

Example



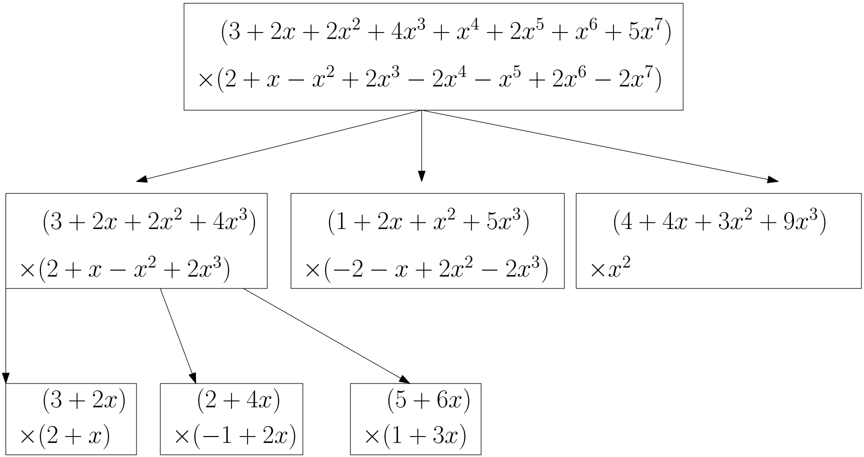
$$(3+2x+2x^2+4x^3+x^4+2x^5+x^6+5x^7)$$
$$\times(2+x-x^2+2x^3-2x^4-x^5+2x^6-2x^7)$$

$$(3+2x+2x^2+4x^3)$$
$$\times(2+x-x^2+2x^3)$$

$$(1+2x+x^2+5x^3)$$
$$\times(-2-x+2x^2-2x^3)$$

$$(4+4x+3x^2+9x^3)$$
$$\times x^2$$

Example



$(3+2x+2x^2+4x^3+x^4+2x^5+x^6+5x^7)$
$\times(2+x-x^2+2x^3-2x^4-x^5+2x^6-2x^7)$

$(3+2x+2x^2+4x^3)$
$\times(2+x-x^2+2x^3)$

$(1+2x+x^2+5x^3)$
$\times(-2-x+2x^2-2x^3)$

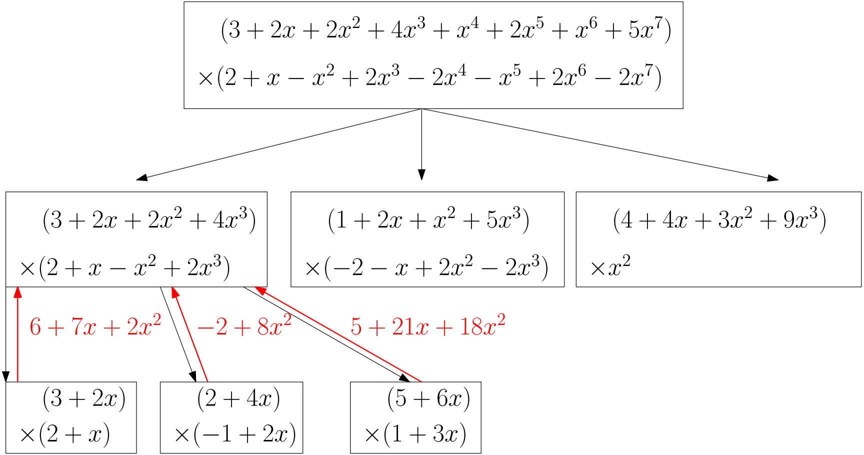
$(4+4x+3x^2+9x^3)$
$\times x^2$

$(3+2x)$
$\times(2+x)$

$(2+4x)$
$\times(-1+2x)$

$(5+6x)$
$\times(1+3x)$

Example



$$(3+2x+2x^2+4x^3+x^4+2x^5+x^6+5x^7)$$
$$\times(2+x-x^2+2x^3-2x^4-x^5+2x^6-2x^7)$$

$$(3+2x+2x^2+4x^3)$$
$$\times(2+x-x^2+2x^3)$$

$$(1+2x+x^2+5x^3)$$
$$\times(-2-x+2x^2-2x^3)$$

$$(4+4x+3x^2+9x^3)$$
$$\times x^2$$

$6+7x+2x^2$

$-2+8x^2$

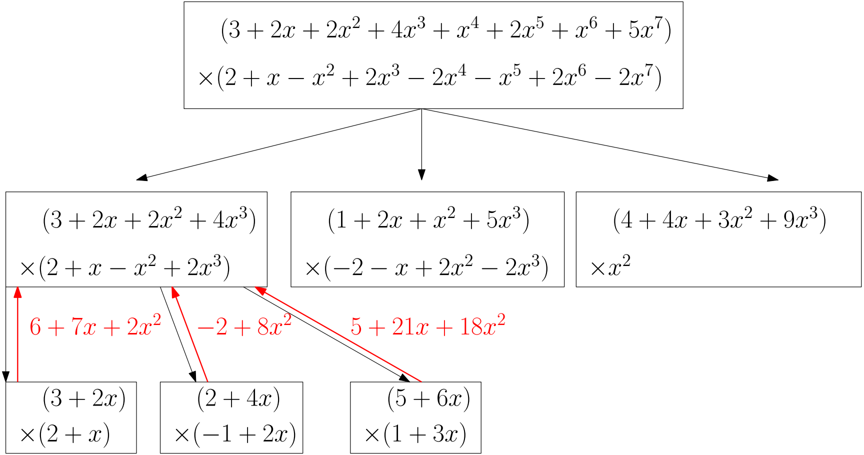
$5+21x+18x^2$

$$(3+2x)$$
$$\times(2+x)$$

$$(2+4x)$$
$$\times(-1+2x)$$

$$(5+6x)$$
$$\times(1+3x)$$

Example



$(3 + 2x + 2x^2 + 4x^3 + x^4 + 2x^5 + x^6 + 5x^7)$
$\times(2 + x - x^2 + 2x^3 - 2x^4 - x^5 + 2x^6 - 2x^7)$

$(3 + 2x + 2x^2 + 4x^3)$
$\times(2 + x - x^2 + 2x^3)$

$(1 + 2x + x^2 + 5x^3)$
$\times(-2 - x + 2x^2 - 2x^3)$

$(4 + 4x + 3x^2 + 9x^3)$
$\times x^2$

$6 + 7x + 2x^2$

$-2 + 8x^2$

$5 + 21x + 18x^2$

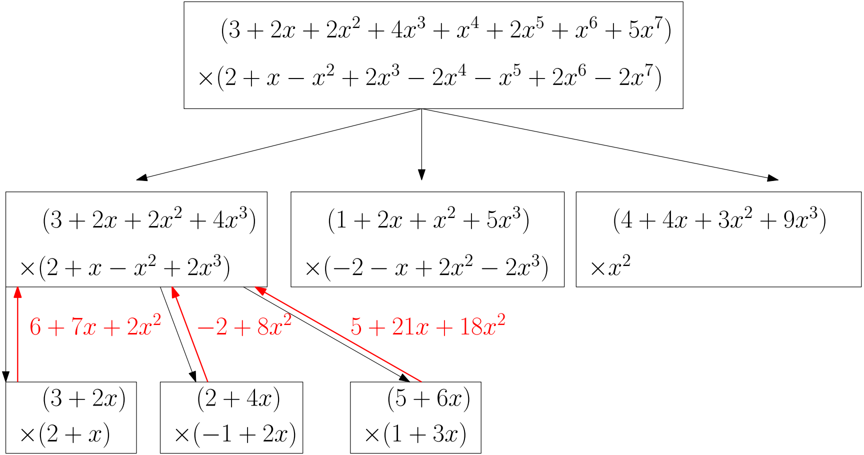
$(3 + 2x)$
$\times(2 + x)$

$(2 + 4x)$
$\times(-1 + 2x)$

$(5 + 6x)$
$\times(1 + 3x)$

$$(5 + 21x + 18x^2) - (6 + 7x + 2x^2) - (-2 + 8x^2) = 1 + 14x + 8x^2$$

Example



$$(3+2x+2x^2+4x^3+x^4+2x^5+x^6+5x^7)$$
$$\times(2+x-x^2+2x^3-2x^4-x^5+2x^6-2x^7)$$

$$(3+2x+2x^2+4x^3)$$
$$\times(2+x-x^2+2x^3)$$

$$(1+2x+x^2+5x^3)$$
$$\times(-2-x+2x^2-2x^3)$$

$$(4+4x+3x^2+9x^3)$$
$$\times x^2$$

$6+7x+2x^2$

$-2+8x^2$

$5+21x+18x^2$

$$(3+2x)$$
$$\times(2+x)$$

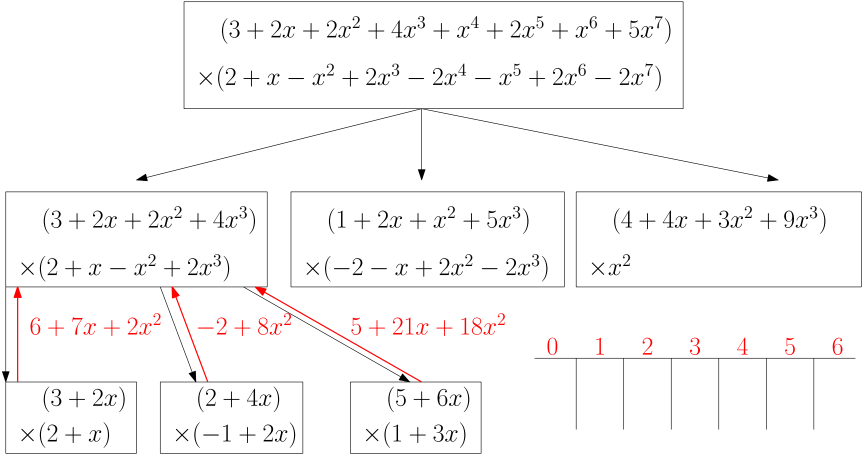
$$(2+4x)$$
$$\times(-1+2x)$$

$$(5+6x)$$
$$\times(1+3x)$$

$$(5+21x+18x^2)-(6+7x+2x^2)-(-2+8x^2)=1+14x+8x^2$$

$$(6+7x+2x^2)+(1+14x+8x^2)x^2+(-2+8x^2)x^4$$
$$=6+7x+3x^2+14x^3+6x^4+8x^6$$

Example



$$(3+2x+2x^2+4x^3+x^4+2x^5+x^6+5x^7)$$
$$\times(2+x-x^2+2x^3-2x^4-x^5+2x^6-2x^7)$$

$$(3+2x+2x^2+4x^3)$$
$$\times(2+x-x^2+2x^3)$$

$$(1+2x+x^2+5x^3)$$
$$\times(-2-x+2x^2-2x^3)$$

$$(4+4x+3x^2+9x^3)$$
$$\times x^2$$

$$(3+2x)$$
$$\times(2+x)$$

$$(2+4x)$$
$$\times(-1+2x)$$

$$(5+6x)$$
$$\times(1+3x)$$

$6+7x+2x^2$

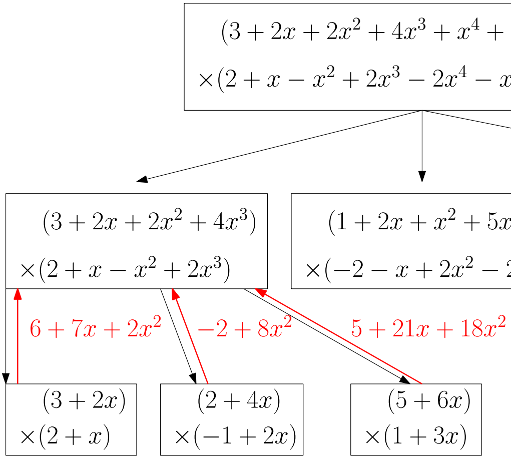
$-2+8x^2$

$5+21x+18x^2$

$$(5+21x+18x^2)-(6+7x+2x^2)-(-2+8x^2)=1+14x+8x^2$$

$$(6+7x+2x^2)+(1+14x+8x^2)x^2+(-2+8x^2)x^4$$
$$=6+7x+3x^2+14x^3+6x^4+8x^6$$

Example



$(3 + 2x + 2x^2 + 4x^3 + x^4 + 2x^5 + x^6 + 5x^7)$
$\times(2 + x - x^2 + 2x^3 - 2x^4 - x^5 + 2x^6 - 2x^7)$

$(3 + 2x + 2x^2 + 4x^3)$
$\times(2 + x - x^2 + 2x^3)$

$(1 + 2x + x^2 + 5x^3)$
$\times(-2 - x + 2x^2 - 2x^3)$

$(4 + 4x + 3x^2 + 9x^3)$
$\times x^2$

$6 + 7x + 2x^2$

$-2 + 8x^2$

$5 + 21x + 18x^2$

$(3 + 2x)$
$\times(2 + x)$

$(2 + 4x)$
$\times(-1 + 2x)$

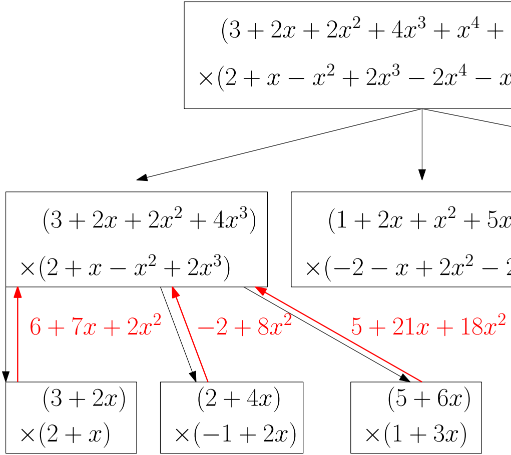
$(5 + 6x)$
$\times(1 + 3x)$

0	1	2	3	4	5	6
6	7	2				

$$(5 + 21x + 18x^2) - (6 + 7x + 2x^2) - (-2 + 8x^2) = 1 + 14x + 8x^2$$

$$\begin{aligned} &(6 + 7x + 2x^2) + (1 + 14x + 8x^2)x^2 + (-2 + 8x^2)x^4 \\ &= 6 + 7x + 3x^2 + 14x^3 + 6x^4 + 8x^6 \end{aligned}$$

Example

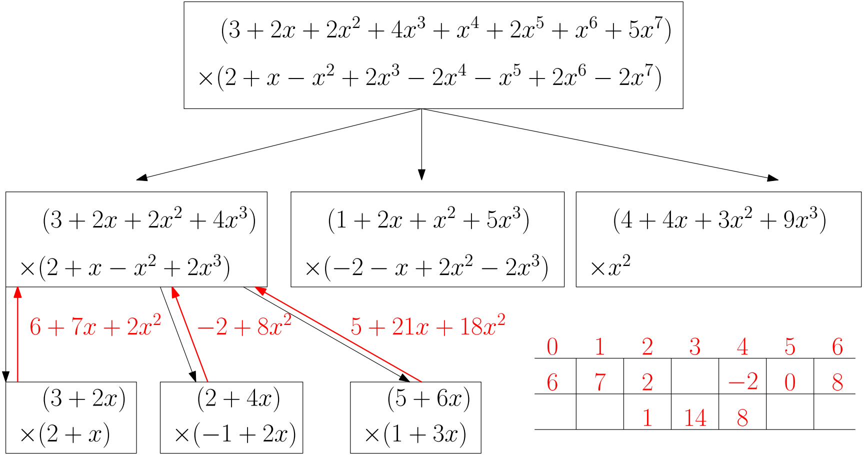


0	1	2	3	4	5	6
6	7	2		-2	0	8

$$(5+21x+18x^2)-(6+7x+2x^2)-(-2+8x^2)=1+14x+8x^2$$

$$\begin{aligned}&(6+7x+2x^2)+(1+14x+8x^2)x^2+(-2+8x^2)x^4\\=&\ 6+7x+3x^2+14x^3+6x^4+8x^6\end{aligned}$$

Example



$(3+2x+2x^2+4x^3+x^4+2x^5+x^6+5x^7)$
$\times(2+x-x^2+2x^3-2x^4-x^5+2x^6-2x^7)$

$(3+2x+2x^2+4x^3)$
$\times(2+x-x^2+2x^3)$

$(1+2x+x^2+5x^3)$
$\times(-2-x+2x^2-2x^3)$

$(4+4x+3x^2+9x^3)$
$\times x^2$

$6+7x+2x^2$

$-2+8x^2$

$5+21x+18x^2$

$(3+2x)$
$\times(2+x)$

$(2+4x)$
$\times(-1+2x)$

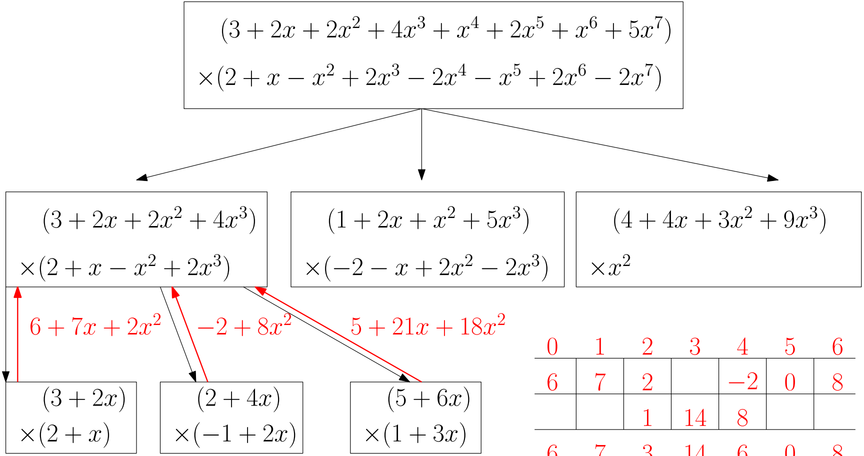
$(5+6x)$
$\times(1+3x)$

0	1	2	3	4	5	6
6	7	2		−2	0	8
		1	14	8		

$$(5+21x+18x^2)-(6+7x+2x^2)-(-2+8x^2)=1+14x+8x^2$$

$$(6+7x+2x^2)+(1+14x+8x^2)x^2+(-2+8x^2)x^4$$
$$=6+7x+3x^2+14x^3+6x^4+8x^6$$

Example



$(3+2x+2x^2+4x^3+x^4+2x^5+x^6+5x^7)$
$\times(2+x-x^2+2x^3-2x^4-x^5+2x^6-2x^7)$

$(3+2x+2x^2+4x^3)$
$\times(2+x-x^2+2x^3)$

$(1+2x+x^2+5x^3)$
$\times(-2-x+2x^2-2x^3)$

$(4+4x+3x^2+9x^3)$
$\times x^2$

$6+7x+2x^2$

$-2+8x^2$

$5+21x+18x^2$

$(3+2x)$
$\times(2+x)$

$(2+4x)$
$\times(-1+2x)$

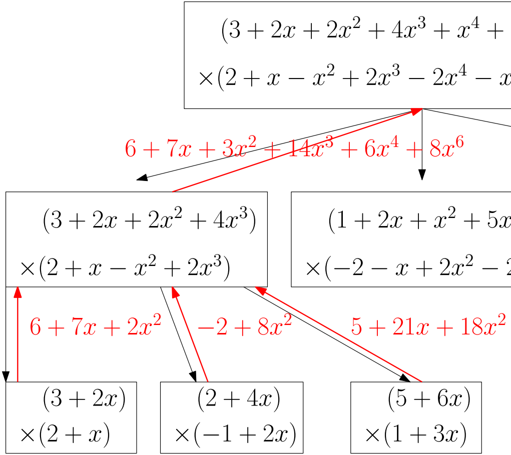
$(5+6x)$
$\times(1+3x)$

0	1	2	3	4	5	6
6	7	2		−2	0	8
		1	14	8		
6	7	3	14	6	0	8

$$(5+21x+18x^2)-(6+7x+2x^2)-(-2+8x^2)=1+14x+8x^2$$

$$(6+7x+2x^2)+(1+14x+8x^2)x^2+(-2+8x^2)x^4$$
$$=6+7x+3x^2+14x^3+6x^4+8x^6$$

Example



$(3+2x+2x^2+4x^3+x^4+2x^5+x^6+5x^7)$
$\times(2+x-x^2+2x^3-2x^4-x^5+2x^6-2x^7)$

$6+7x+3x^2+14x^3+6x^4+8x^6$

$(3+2x+2x^2+4x^3)$
$\times(2+x-x^2+2x^3)$

$(1+2x+x^2+5x^3)$
$\times(-2-x+2x^2-2x^3)$

$(4+4x+3x^2+9x^3)$
$\times x^2$

$6+7x+2x^2$

$-2+8x^2$

$5+21x+18x^2$

$(3+2x)$
$\times(2+x)$

$(2+4x)$
$\times(-1+2x)$

$(5+6x)$
$\times(1+3x)$

0	1	2	3	4	5	6
6	7	2		−2	0	8
		1	14	8		
6	7	3	14	6	0	8

$$(5+21x+18x^2)-(6+7x+2x^2)-(-2+8x^2)=1+14x+8x^2$$

$$(6+7x+2x^2)+(1+14x+8x^2)x^2+(-2+8x^2)x^4$$
$$=6+7x+3x^2+14x^3+6x^4+8x^6$$

Outline

1. Divide-and-Conquer
2. Counting Inversions
3. Solving Recurrences
4. Quicksort and Selection
    - Quicksort
    - Lower Bound for Comparison-Based Sorting Algorithms
    - Selection Problem
5. Polynomial Multiplication
6. Strassen's Algorithm for Matrix Multiplication
7. FFT(Fast Fourier Transform): Polynomial Multiplication in $O(n \log n)$ Time
8. Finding Closest Pair of Points in 2D Euclidean Space
9. Computing n-th Fibonacci Number

Matrix Multiplication

Input: two $n \times n$ matrices A and B

Output: $C = AB$

Matrix Multiplication

Input: two $n \times n$ matrices A and B

Output: $C = AB$

Naive Algorithm: matrix-multiplication(A, B, n)

```
for $i \leftarrow 1$ to $n$ do
    for $j \leftarrow 1$ to $n$ do
        $C[i,j] \leftarrow 0$
        for $k \leftarrow 1$ to $n$ do
            $C[i,j] \leftarrow C[i,j] + A[i,k] \times B[k,j]$
return $C$
```

Matrix Multiplication

Input: two $n \times n$ matrices A and B

Output: $C = AB$

Naive Algorithm: matrix-multiplication(A, B, n)

```
for $i \leftarrow 1$ to $n$ do
    for $j \leftarrow 1$ to $n$ do
        $C[i,j] \leftarrow 0$
        for $k \leftarrow 1$ to $n$ do
            $C[i,j] \leftarrow C[i,j] + A[i,k] \times B[k,j]$
return $C$
```

- running time $= O(n^3)$

Try to Use Divide-and-Conquer

$$A = \begin{array}{|c|c|l} \multicolumn{1}{c}{} & \multicolumn{1}{c}{\overbrace{\hphantom{A_{12}}}^{n/2}} & \\ \hline A_{11} & A_{12} & \left.\vphantom{A_{12}}\right\} n/2 \\ \hline A_{21} & A_{22} & \\ \hline \end{array} \qquad B = \begin{array}{|c|c|l} \multicolumn{1}{c}{} & \multicolumn{1}{c}{\overbrace{\hphantom{B_{12}}}^{n/2}} & \\ \hline B_{11} & B_{12} & \left.\vphantom{B_{12}}\right\} n/2 \\ \hline B_{21} & B_{22} & \\ \hline \end{array}$$

- $C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$
- matrix_multiplication(A, B) recursively calls matrix_multiplication(A_{11}, B_{11}), matrix_multiplication(A_{12}, B_{21}), $\cdots$

Try to Use Divide-and-Conquer

$$A = \begin{array}{|c|c|l} \multicolumn{1}{c}{} & \multicolumn{1}{c}{\overbrace{\phantom{A_{12}}}^{n/2}} & \\ \hline A_{11} & A_{12} & \}n/2 \\ \hline A_{21} & A_{22} & \\ \hline \end{array} \qquad B = \begin{array}{|c|c|l} \multicolumn{1}{c}{} & \multicolumn{1}{c}{\overbrace{\phantom{B_{12}}}^{n/2}} & \\ \hline B_{11} & B_{12} & \}n/2 \\ \hline B_{21} & B_{22} & \\ \hline \end{array}$$

- $C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$
- matrix_multiplication(A, B) recursively calls matrix_multiplication(A_{11}, B_{11}), matrix_multiplication(A_{12}, B_{21}), $\cdots$
- Recurrence for running time: $T(n) = 8T(n/2) + O(n^2)$
- $T(n) = O(n^3)$

Try to Use Divide-and-Conquer

$$A = \begin{array}{|c|c|l} \multicolumn{1}{c}{} & \multicolumn{1}{c}{\overbrace{\hphantom{A_{12}}}^{n/2}} & \\ \hline A_{11} & A_{12} & \left.\vphantom{A_{12}}\right\} n/2 \\ \hline A_{21} & A_{22} & \\ \hline \end{array} \qquad B = \begin{array}{|c|c|l} \multicolumn{1}{c}{} & \multicolumn{1}{c}{\overbrace{\hphantom{B_{12}}}^{n/2}} & \\ \hline B_{11} & B_{12} & \left.\vphantom{B_{12}}\right\} n/2 \\ \hline B_{21} & B_{22} & \\ \hline \end{array}$$

- $C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$
- matrix_multiplication(A, B) recursively calls matrix_multiplication(A_{11}, B_{11}), matrix_multiplication(A_{12}, B_{21}), $\cdots$
- Recurrence for running time: $T(n) = 8T(n/2) + O(n^2)$
- $T(n) = O(n^3)$
- Strassen's Algorithm: $T(n) = {\color{red}7}T(n/2) + O(n^2)$

Try to Use Divide-and-Conquer

$$A = \begin{array}{|c|c|l} \multicolumn{1}{c}{} & \multicolumn{1}{c}{\overbrace{\hphantom{A_{12}}}^{n/2}} & \\ \cline{1-2} A_{11} & A_{12} & \}n/2 \\ \cline{1-2} A_{21} & A_{22} & \\ \cline{1-2} \end{array} \qquad B = \begin{array}{|c|c|l} \multicolumn{1}{c}{} & \multicolumn{1}{c}{\overbrace{\hphantom{B_{12}}}^{n/2}} & \\ \cline{1-2} B_{11} & B_{12} & \}n/2 \\ \cline{1-2} B_{21} & B_{22} & \\ \cline{1-2} \end{array}$$

- $C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$
- matrix_multiplication(A, B) recursively calls matrix_multiplication(A_{11}, B_{11}), matrix_multiplication(A_{12}, B_{21}), $\cdots$
- Recurrence for running time: $T(n) = 8T(n/2) + O(n^2)$
- $T(n) = O(n^3)$
- Strassen's Algorithm: $T(n) = 7T(n/2) + O(n^2)$
- Solving Recurrence $T(n) = O(n^{\log_2 7}) = O(n^{2.808})$

Strassen's Algorithm

$$A = \begin{array}{|c|c|l} \multicolumn{1}{c}{} & \multicolumn{1}{c}{\overbrace{\hphantom{A_{12}}}^{n/2}} & \\ \hline A_{11} & A_{12} & \big\} n/2 \\ \hline A_{21} & A_{22} & \\ \hline \end{array} \qquad B = \begin{array}{|c|c|l} \multicolumn{1}{c}{} & \multicolumn{1}{c}{\overbrace{\hphantom{B_{12}}}^{n/2}} & \\ \hline B_{11} & B_{12} & \big\} n/2 \\ \hline B_{21} & B_{22} & \\ \hline \end{array}$$

- $C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$

Strassen's Algorithm

$$A = \begin{array}{|c|c|}\hline A_{11} & \overbrace{A_{12}}^{n/2} \Big\} n/2 \\ \hline A_{21} & A_{22} \\ \hline \end{array} \qquad B = \begin{array}{|c|c|}\hline B_{11} & \overbrace{B_{12}}^{n/2} \Big\} n/2 \\ \hline B_{21} & B_{22} \\ \hline \end{array}$$

- $C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$

- $M_1 \leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22})$
- $M_2 \leftarrow (A_{21} + A_{22}) \times B_{11}$
- $M_3 \leftarrow A_{11} \times (B_{12} - B_{22})$
- $M_4 \leftarrow A_{22} \times (B_{21} - B_{11})$
- $M_5 \leftarrow (A_{11} + A_{12}) \times B_{22}$
- $M_6 \leftarrow (A_{21} - A_{11}) \times (B_{11} + B_{12})$
- $M_7 \leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22})$

- $C_{11} \leftarrow M_1 + M_4 - M_5 + M_7$
- $C_{12} \leftarrow M_3 + M_5$
- $C_{21} \leftarrow M_2 + M_4$
- $C_{22} \leftarrow M_1 - M_2 + M_3 + M_6$

Outline

1. Divide-and-Conquer
2. Counting Inversions
3. Solving Recurrences
4. Quicksort and Selection
    - Quicksort
    - Lower Bound for Comparison-Based Sorting Algorithms
    - Selection Problem
5. Polynomial Multiplication
6. Strassen's Algorithm for Matrix Multiplication
7. FFT(Fast Fourier Transform): Polynomial Multiplication in $O(n \log n)$ Time
8. Finding Closest Pair of Points in 2D Euclidean Space
9. Computing n-th Fibonacci Number

Interpolation of Polynomials

- $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$

Interpolation of Polynomials

- $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$
- Known: given the value of $p(x)$ for n different values of x, p is uniquely determined

Interpolation of Polynomials

- $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$
- Known: given the value of $p(x)$ for n different values of x, p is uniquely determined
- $p(x) = 1 - x + 2x^2 : p(0) = 1, p(1) = 2, p(2) = 7$.

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 7 \end{pmatrix}$$

Interpolation of Polynomials

- $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$
- Known: given the value of $p(x)$ for n different values of x, p is uniquely determined
- $p(x) = 1 - x + 2x^2 : p(0) = 1, p(1) = 2, p(2) = 7.$

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 7 \end{pmatrix}$$

- Given $p(0) = 1, p(1) = 2, p(2) = 7$, to recover p:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ 2 \\ 7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{3}{2} & 2 & -\frac{1}{2} \\ \frac{1}{2} & -1 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 7 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}$$

Interpolation of Polynomials

- $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$
- Known: given the value of $p(x)$ for n different values of x, p is uniquely determined
- $p(x) = 1 - x + 2x^2$: $p(0) = 1, p(1) = 2, p(2) = 7$.

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 7 \end{pmatrix}$$

- Given $p(0) = 1, p(1) = 2, p(2) = 7$, to recover p:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ 2 \\ 7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{3}{2} & 2 & -\frac{1}{2} \\ \frac{1}{2} & -1 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 7 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}$$

- $p(x) = 1 - x + 2x^2$

Using Interpolation for Polynomial Multiplication

Using Interpolation for Polynomial Multiplication

- $p(x) = 1 - x + 2x^2, \qquad q(x) = 3 - x^2$

Using Interpolation for Polynomial Multiplication

- $p(x) = 1 - x + 2x^2, \qquad q(x) = 3 - x^2$
- Interpolation on 5 points $\{0, 1, 2, 3, 4\}$:

$$\text{interpolation for } p: \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 2 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 7 \\ 16 \\ 29 \end{pmatrix}$$

$$\text{interpolation for } q: \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \end{pmatrix} \begin{pmatrix} 3 \\ 0 \\ -1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ -1 \\ -6 \\ -13 \end{pmatrix}$$

Interpolation of pq:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ -7 \\ -102 \\ -377 \end{pmatrix}$$

Interpolation of pq:

$$\begin{pmatrix}1 & 0 & 0 & 0 & 0\\ 1 & 1 & 1 & 1 & 1\\ 1 & 2 & 4 & 8 & 16\\ 1 & 3 & 9 & 27 & 81\\ 1 & 4 & 16 & 64 & 256\end{pmatrix}\begin{pmatrix}c_0\\ c_1\\ c_2\\ c_3\\ c_4\end{pmatrix} = \begin{pmatrix}3\\ 4\\ -7\\ -102\\ -377\end{pmatrix}$$

$$\begin{pmatrix}c_0\\ c_1\\ c_2\\ c_3\\ c_4\end{pmatrix} = \begin{pmatrix}1 & 0 & 0 & 0 & 0\\ 1 & 1 & 1 & 1 & 1\\ 1 & 2 & 4 & 8 & 16\\ 1 & 3 & 9 & 27 & 81\\ 1 & 4 & 16 & 64 & 256\end{pmatrix}^{-1}\begin{pmatrix}3\\ 4\\ -7\\ -96\\ -377\end{pmatrix}$$

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -\frac{25}{12} & 4 & -3 & \frac{4}{3} & -\frac{1}{4} \\ \frac{35}{24} & -\frac{13}{3} & \frac{19}{4} & -\frac{7}{3} & \frac{11}{24} \\ -\frac{5}{12} & \frac{3}{2} & -2 & \frac{7}{6} & -\frac{1}{4} \\ \frac{1}{24} & -\frac{1}{6} & \frac{1}{4} & -\frac{1}{6} & \frac{1}{24} \end{pmatrix} \begin{pmatrix} 3 \\ 4 \\ -7 \\ -96 \\ -377 \end{pmatrix} = \begin{pmatrix} 3 \\ -3 \\ 5 \\ 1 \\ -2 \end{pmatrix}$$

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -\frac{25}{12} & 4 & -3 & \frac{4}{3} & -\frac{1}{4} \\ \frac{35}{24} & -\frac{13}{3} & \frac{19}{4} & -\frac{7}{3} & \frac{11}{24} \\ -\frac{5}{12} & \frac{3}{2} & -2 & \frac{7}{6} & -\frac{1}{4} \\ \frac{1}{24} & -\frac{1}{6} & \frac{1}{4} & -\frac{1}{6} & \frac{1}{24} \end{pmatrix} \begin{pmatrix} 3 \\ 4 \\ -7 \\ -96 \\ -377 \end{pmatrix} = \begin{pmatrix} 3 \\ -3 \\ 5 \\ 1 \\ -2 \end{pmatrix}$$

$$pq = (1 - x + 2x^2)(3 - x^2) = 3 - 3x + 5x^2 + x^3 - 2x^4$$

Multiplication of two polynomials of degree $n-1$

- Choose $2n-1$ distinct values $x_0, x_1, x_2, \cdots, x_{m-1}$ carefully, $m = 2n-1$
- Compute the interpolation of p and q:

$$M := \begin{pmatrix} 1 & x_0 & x_0^2 & x_0^3 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & x_2^3 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{m-1} & x_{m-1}^2 & x_m^3 & \cdots & x_{m-1}^{n-1} \end{pmatrix}$$

$$M \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{m-1} \end{pmatrix} \qquad M \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ \vdots \\ z_{m-1} \end{pmatrix}$$

Multiplication of two polynomials of degree $n-1$

$$M \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} y_0 z_0 \\ y_1 z_1 \\ y_2 z_2 \\ \vdots \\ y_{m-1} z_{m-1} \end{pmatrix} \qquad \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = M^{-1} \begin{pmatrix} y_0 z_0 \\ y_1 z_1 \\ y_2 z_2 \\ \vdots \\ y_{m-1} z_{m-1} \end{pmatrix}$$

Multiplication of two polynomials of degree $n-1$

$$M \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} y_0 z_0 \\ y_1 z_1 \\ y_2 z_2 \\ \vdots \\ y_{m-1} z_{m-1} \end{pmatrix} \qquad \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = M^{-1} \begin{pmatrix} y_0 z_0 \\ y_1 z_1 \\ y_2 z_2 \\ \vdots \\ y_{m-1} z_{m-1} \end{pmatrix}$$

$$\begin{aligned} &(a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}) \\ \times\ &(b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}) \\ =\ &(c_0 + c_1 x + c_2 x^2 + \cdots + c_{2n-2} x^{2n-2}) \end{aligned}$$

Q: How should we set $x_0, x_1, \cdots, x_{n-1}$ so that we can compute Ma and $M^{-1}y$ fast (for any $a, y \in \mathbb{R}^{\{0,1,\cdots,n-1\}}$)?

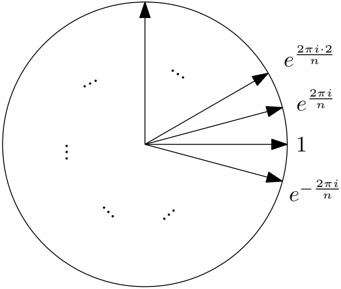
Q: How should we set $x_0, x_1, \cdots, x_{n-1}$ so that we can compute Ma and $M^{-1}y$ fast (for any $a, y \in \mathbb{R}^{\{0,1,\cdots,n-1\}}$)?

A: Use the n complex roots of the equation $x^n = 1$

Q: How should we set $x_0, x_1, \cdots, x_{n-1}$ so that we can compute Ma and $M^{-1}y$ fast (for any $a, y \in \mathbb{R}^{\{0,1,\cdots,n-1\}}$)?

A: Use the n complex roots of the equation $x^n = 1$

- $e^{\frac{2\pi i \cdot k}{n}} = \cos\left(\frac{2\pi \cdot k}{n}\right) + i \cdot \sin\left(\frac{2\pi \cdot k}{n}\right), k \in \{0, 1, \cdots, n-1\}$
- $\omega := e^{\frac{2\pi i}{n}}$, n-th roots are $1, \omega, \omega^2, \cdots, \omega^{n-1}$



$$F_n := \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \end{pmatrix}$$

- Interpolation and Inverse-Interpolation:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = F_n \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} \qquad \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = F_n^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$$F_n := \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \end{pmatrix}$$

- Interpolation and Inverse-Interpolation:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = F_n \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} \qquad \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = F_n^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

- Interpolation: Fast Fourier Transform (FFT)
- Invert-Interpolation: Inverse Fast Fourier Transform (iFFT)

Fast Fourier Transform: Divide and Conquer

- Assume n is even.

Breaking polynomial into even and odd parts

- $p_{\textsf{even}}(x) := a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1}$
- $p_{\textsf{old}}(x) := a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1}$
- $p(x) = p_{\textsf{even}}(x^2) + p_{\textsf{odd}}(x^2) \cdot x$

Fast Fourier Transform: Divide and Conquer

- Assume n is even.

Breaking polynomial into even and odd parts

- $p_{\mathsf{even}}(x) := a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1}$
- $p_{\mathsf{old}}(x) := a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1}$
- $p(x) = p_{\mathsf{even}}(x^2) + p_{\mathsf{odd}}(x^2) \cdot x$

$$\begin{aligned} p(\omega^k) &= p_{\mathsf{even}}(\omega^{2k}) + p_{\mathsf{odd}}(\omega^{2k}) \cdot \omega^k, \qquad & k = 0, 1, \cdots, \frac{n}{2} - 1 \\ p(\omega^{n/2+k}) &= p_{\mathsf{even}}(\omega^{2k}) - p_{\mathsf{odd}}(\omega^{2k}) \cdot \omega^k, \qquad & k = 0, 1, \cdots, \frac{n}{2} - 1 \end{aligned}$$

- Assume n is an integer power of 2

```
FFT$(n, a_0, a_1, \cdots, a_{n-1})$
if $n = 1$ then return $(a_0)$
$(e_0, e_1, \cdots, e_{n/2-1}) \leftarrow \mathrm{FFT}(n/2, a_0, a_2, \cdots, a_{n-2})$
$(o_0, o_1, \cdots, o_{n/2-1}) \leftarrow \mathrm{FFT}(n/2, a_1, a_3, \cdots, a_{n-1})$
for $k \leftarrow 0, 1, 2, \cdots n/2 - 1$ do
    $y_k \leftarrow e_k + o_k \cdot \omega^k$
    $y_{n/2+k} \leftarrow e_k - o_k \cdot \omega^k$
return $(y_0, y_1, \cdots, y_{n-1})$
```

- Assume n is an integer power of 2

FFT$(n, a_0, a_1, \cdots, a_{n-1})$

```
if $n = 1$ then return $(a_0)$
$(e_0, e_1, \cdots, e_{n/2-1}) \leftarrow \mathrm{FFT}(n/2, a_0, a_2, \cdots, a_{n-2})$
$(o_0, o_1, \cdots, o_{n/2-1}) \leftarrow \mathrm{FFT}(n/2, a_1, a_3, \cdots, a_{n-1})$
for $k \leftarrow 0, 1, 2, \cdots n/2 - 1$ do
    $y_k \leftarrow e_k + o_k \cdot \omega^k$
    $y_{n/2+k} \leftarrow e_k - o_k \cdot \omega^k$
return $(y_0, y_1, \cdots, y_{n-1})$
```

- Recurrence for running time: $T(n) = 2T(n/2) + O(n)$
- $T(n) = O(n \log n)$

Example for one recursion of FFT

- $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) = (3, 2, 1, 2, 5, 6, 1, 4)$

$$\begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 5 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 \\ -2 \\ 6 \\ -2 \end{pmatrix}$$

$$\begin{pmatrix} o_0 \\ o_1 \\ o_2 \\ o_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 2 \\ 2 \\ 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 14 \\ -4 - 2i \\ 2 \\ -4 + 2i \end{pmatrix}$$

Example for one recursion of FFT

- $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) = (3, 2, 1, 2, 5, 6, 1, 4)$

$$\begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 5 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 \\ -2 \\ 6 \\ -2 \end{pmatrix}$$

$$\begin{pmatrix} o_0 \\ o_1 \\ o_2 \\ o_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 2 \\ 2 \\ 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 14 \\ -4-2i \\ 2 \\ -4+2i \end{pmatrix}$$

- $\omega = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}$

Example for one recursion of FFT

- $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) = (3, 2, 1, 2, 5, 6, 1, 4)$

$$\begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 5 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 \\ -2 \\ 6 \\ -2 \end{pmatrix}$$

$$\begin{pmatrix} o_0 \\ o_1 \\ o_2 \\ o_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 2 \\ 2 \\ 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 14 \\ -4 - 2i \\ 2 \\ -4 + 2i \end{pmatrix}$$

- $\omega = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}$
- $y_0 = e_0 + o_0 = 10 + 14 = 24$

Example for one recursion of FFT

- $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) = (3, 2, 1, 2, 5, 6, 1, 4)$

$$\begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 5 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 \\ -2 \\ 6 \\ -2 \end{pmatrix}$$

$$\begin{pmatrix} o_0 \\ o_1 \\ o_2 \\ o_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 2 \\ 2 \\ 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 14 \\ -4 - 2i \\ 2 \\ -4 + 2i \end{pmatrix}$$

- $\omega = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}$
- $y_0 = e_0 + o_0 = 10 + 14 = 24$
- $y_1 = e_1 + o_1\omega = -2 + (-4 - 2i)\left(\frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}\right) = -2 - 2\sqrt{2} - 3\sqrt{2}i$

Example for one recursion of FFT

- $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) = (3, 2, 1, 2, 5, 6, 1, 4)$

$$\begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 5 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 \\ -2 \\ 6 \\ -2 \end{pmatrix}$$

$$\begin{pmatrix} o_0 \\ o_1 \\ o_2 \\ o_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 2 \\ 2 \\ 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 14 \\ -4-2i \\ 2 \\ -4+2i \end{pmatrix}$$

- $\omega = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}$
- $y_0 = e_0 + o_0 = 10 + 14 = 24$
- $y_1 = e_1 + o_1\omega = -2 + (-4 - 2i)\left(\frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}\right) = -2 - 2\sqrt{2} - 3\sqrt{2}i$
- $y_6 = e_2 - o_2\omega^2 = 6 - 2i$

Example for one recursion of FFT

- $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) = (3, 2, 1, 2, 5, 6, 1, 4)$

$$\begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 5 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 \\ -2 \\ 6 \\ -2 \end{pmatrix}$$

$$\begin{pmatrix} o_0 \\ o_1 \\ o_2 \\ o_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 2 \\ 2 \\ 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 14 \\ -4-2i \\ 2 \\ -4+2i \end{pmatrix}$$

- $\omega = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}$
- $y_0 = e_0 + o_0 = 10 + 14 = 24$
- $y_1 = e_1 + o_1\omega = -2 + (-4 - 2i)\left(\frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}\right) = -2 - 2\sqrt{2} - 3\sqrt{2}i$
- $y_6 = e_2 - o_2\omega^2 = 6 - 2i \qquad y_7 = e_3 - o_3\omega^3$

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$
$$q(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$$

multiplying p and q, ▷ assuming n is a power of 2

```
1: $y \leftarrow \mathrm{FFT}(2n, a_0, a_1, \cdots, a_{n-1}, 0, 0, \cdots, 0)$
2: $z \leftarrow \mathrm{FFT}(2n, b_0, b_1, \cdots, b_{n-1}, 0, 0, \cdots, 0)$
3: $c \leftarrow \mathrm{iFFT}(2n, y_0 z_0, y_1 z_1, \cdots, y_{2n-1} z_{2n-1})$
4: return $(c_0, c_1, \cdots, c_{2n-2})$
```

- iFFT$(n, y_0, y_1, \cdots, y_{n-1})$: inverse FFT procedure: multiplying input vector y by the inverse of F_n, which is

$$\frac{1}{n}\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \end{pmatrix}$$

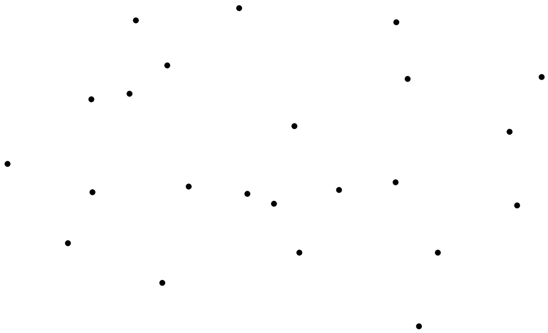
Outline

1. Divide-and-Conquer
2. Counting Inversions
3. Solving Recurrences
4. Quicksort and Selection
    - Quicksort
    - Lower Bound for Comparison-Based Sorting Algorithms
    - Selection Problem
5. Polynomial Multiplication
6. Strassen's Algorithm for Matrix Multiplication
7. FFT(Fast Fourier Transform): Polynomial Multiplication in $O(n \log n)$ Time
8. **Finding Closest Pair of Points in 2D Euclidean Space**
9. Computing n-th Fibonacci Number

Closest Pair

Input: n points in plane: $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$

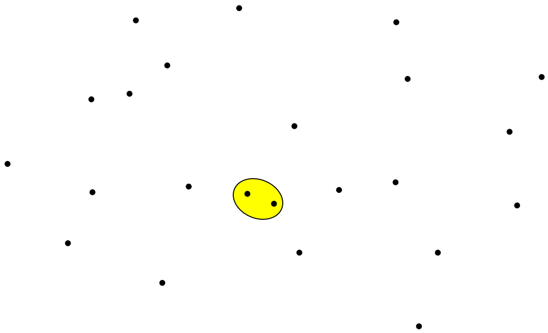
Output: the pair of points that are closest



Closest Pair

Input: n points in plane: $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$

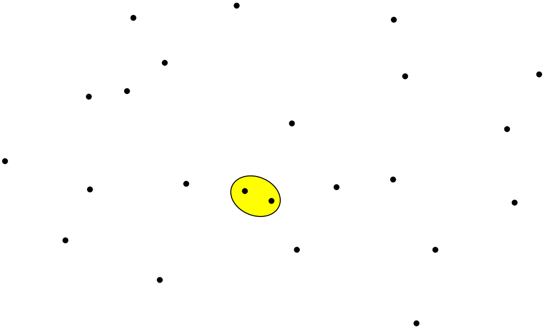
Output: the pair of points that are closest



Closest Pair

Input: n points in plane: $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$

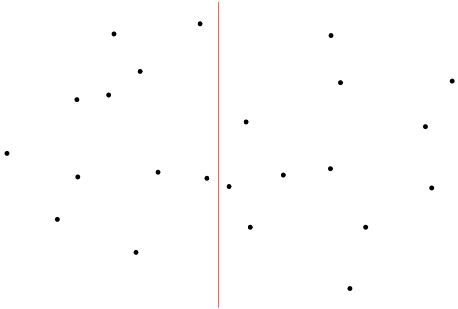
Output: the pair of points that are closest



- Trivial algorithm: $O(n^2)$ running time

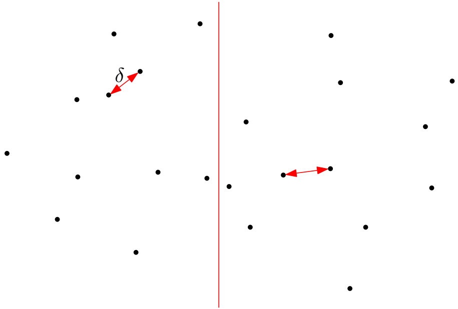
Divide-and-Conquer Algorithm for Closest Pair

- **Divide**: Divide the points into two halves via a vertical line



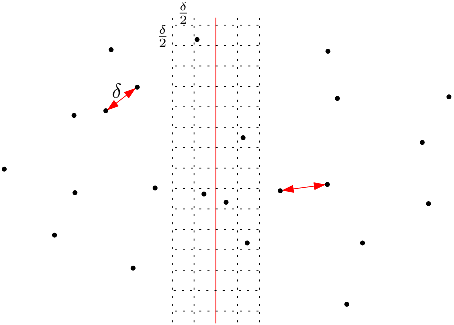
Divide-and-Conquer Algorithm for Closest Pair

- **Divide**: Divide the points into two halves via a vertical line
- **Conquer**: Solve two sub-instances recursively

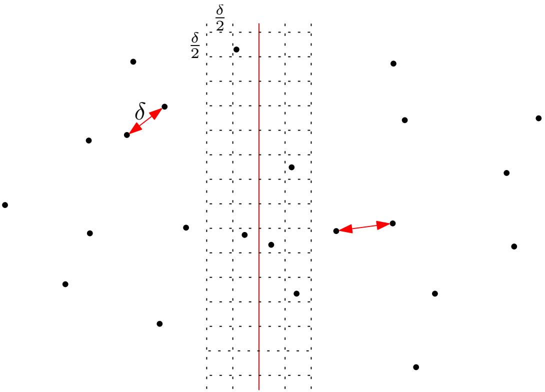


Divide-and-Conquer Algorithm for Closest Pair

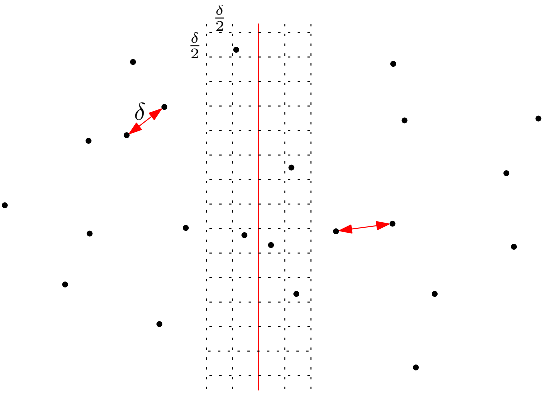
- **Divide**: Divide the points into two halves via a vertical line
- **Conquer**: Solve two sub-instances recursively
- **Combine**: Check if there is a closer pair between left-half and right-half



Divide-and-Conquer Algorithm for Closest Pair

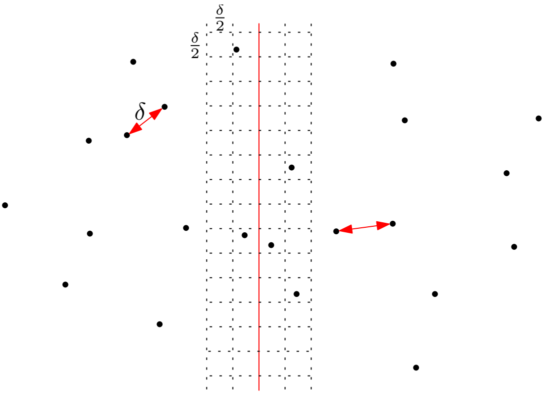


Divide-and-Conquer Algorithm for Closest Pair



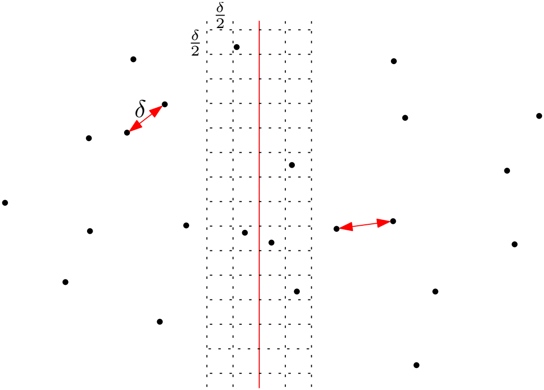
- Each box contains at most one pair

Divide-and-Conquer Algorithm for Closest Pair



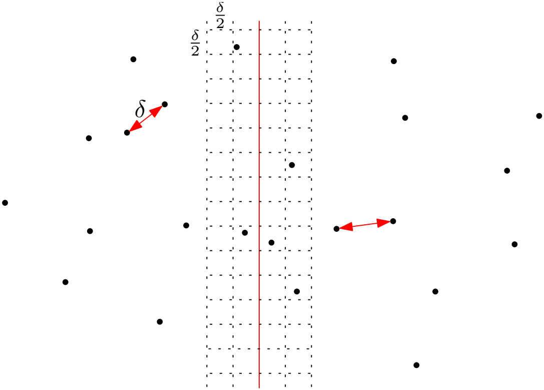
- Each box contains at most one pair
- For each point, only need to consider $O(1)$ boxes nearby

Divide-and-Conquer Algorithm for Closest Pair



- Each box contains at most one pair
- For each point, only need to consider $O(1)$ boxes nearby
- Implementation: Sort points inside the stripe according to y-coordinates

Divide-and-Conquer Algorithm for Closest Pair



- Each box contains at most one pair
- For each point, only need to consider $O(1)$ boxes nearby
- Implementation: Sort points inside the stripe according to y-coordinates
- For every point, consider $O(1)$ points around it in the order

- time for combine step $= O(n \log n)$
- recurrence: $T(n) = 2T(n/2) + O(n \log n)$

- time for combine step $= O(n \log n)$
- recurrence: $T(n) = 2T(n/2) + O(n \log n)$
- solving recurrence: $T(n) = ?$

- time for combine step $= O(n \log n)$
- recurrence: $T(n) = 2T(n/2) + O(n \log n)$
- solving recurrence: $T(n) = O(n \log^2 n)$

- time for combine step $= O(n \log n)$
- recurrence: $T(n) = 2T(n/2) + O(n \log n)$
- solving recurrence: $T(n) = O(n \log^2 n)$

Improve the running time of combine step to $O(n)$

- also sort the points in ascending order of y values at the beginning
- pass the sequence to the root recursion
- constructing two sub-sequences from the sequence, and pass them to the two sub-recursions respectively

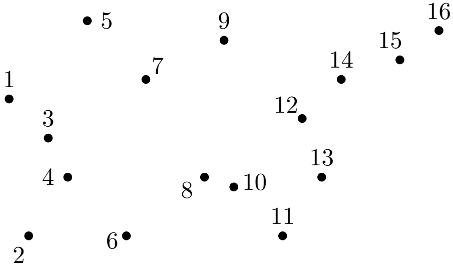
- time for combine step $= O(n \log n)$
- recurrence: $T(n) = 2T(n/2) + O(n \log n)$
- solving recurrence: $T(n) = O(n \log^2 n)$

Improve the running time of combine step to $O(n)$

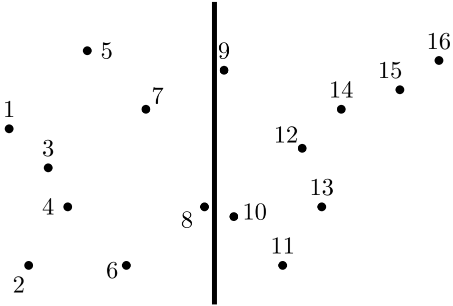
- also sort the points in ascending order of y values at the beginning
- pass the sequence to the root recursion
- constructing two sub-sequences from the sequence, and pass them to the two sub-recursions respectively

- $T(n) = 2T(n/2) + O(n) \Longrightarrow T(n) = O(n \log n)$

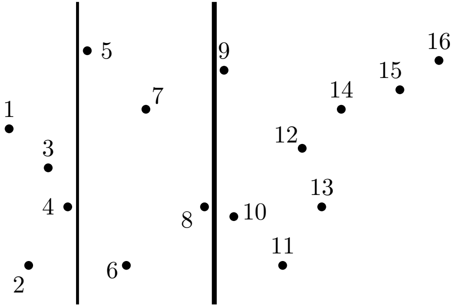
Example for Closest Pair



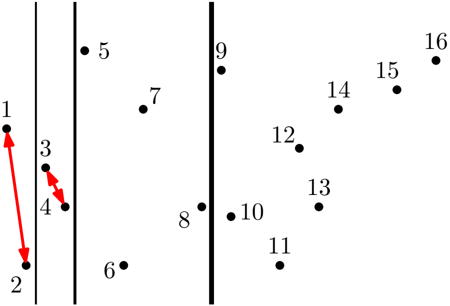
Example for Closest Pair



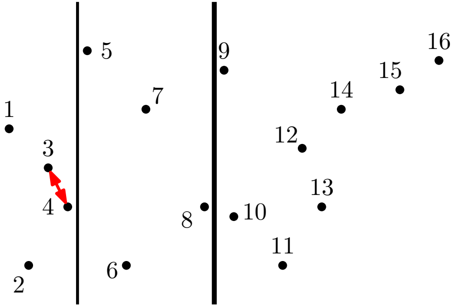
Example for Closest Pair



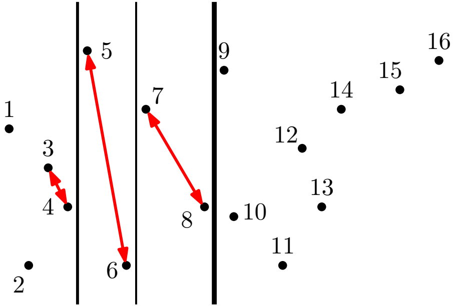
Example for Closest Pair



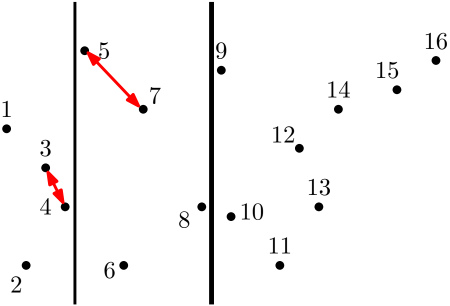
Example for Closest Pair



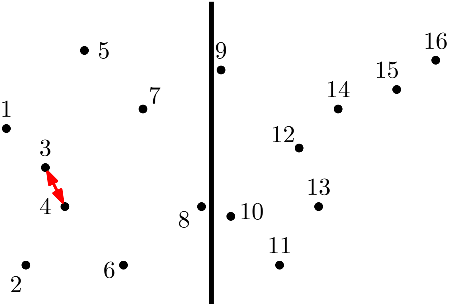
Example for Closest Pair



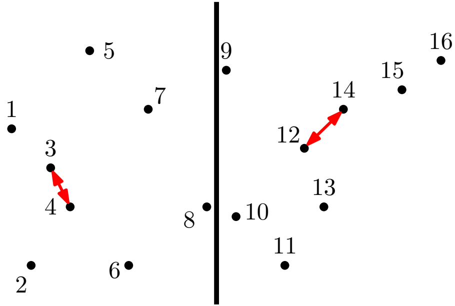
Example for Closest Pair



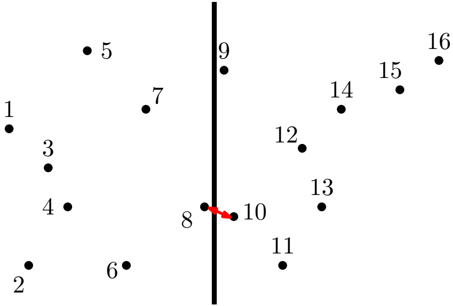
Example for Closest Pair



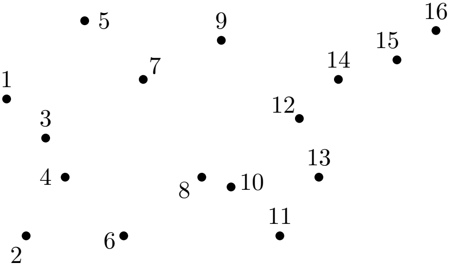
Example for Closest Pair



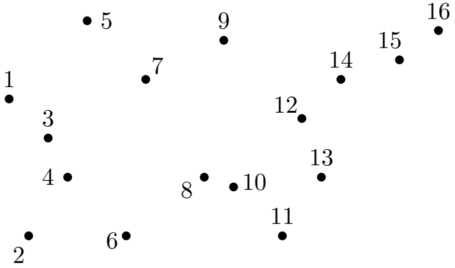
Example for Closest Pair



Example for Closest Pair

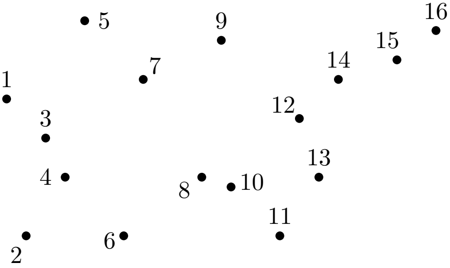


Example for Closest Pair



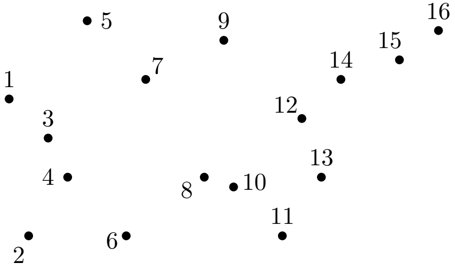
- CP$(1, 16, (5, 16, 9, 15, 7, 14, 1, 12, 3, 4, 8, 13, 10, 11, 2, 6))$

Example for Closest Pair



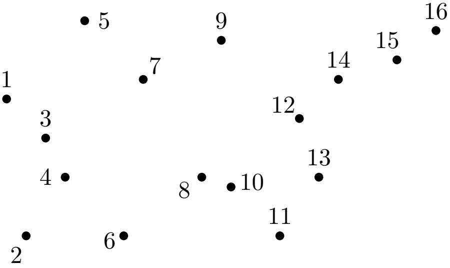
- CP$(1, 16, (5, 16, 9, 15, 7, 14, 1, 12, 3, 4, 8, 13, 10, 11, 2, 6))$
- CP$(1, 8, (5, 7, 1, 3, 4, 8, 2, 6))$

Example for Closest Pair



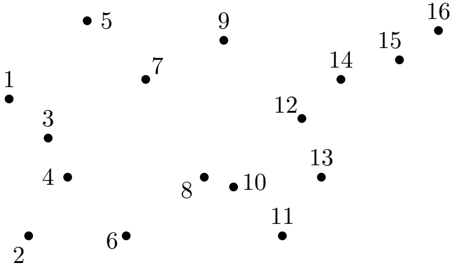
- CP$(1, 16, (5, 16, 9, 15, 7, 14, 1, 12, 3, 4, 8, 13, 10, 11, 2, 6))$
- CP$(1, 8, (5, 7, 1, 3, 4, 8, 2, 6))$
- CP$(1, 4, (1, 3, 4, 2))$

Example for Closest Pair



- CP$(1, 16, (5, 16, 9, 15, 7, 14, 1, 12, 3, 4, 8, 13, 10, 11, 2, 6))$
-     CP$(1, 8, (5, 7, 1, 3, 4, 8, 2, 6))$
-         CP$(1, 4, (1, 3, 4, 2))$
-         CP$(5, 8, (5, 7, 8, 6))$

Example for Closest Pair



- CP$(1, 16, (5, 16, 9, 15, 7, 14, 1, 12, 3, 4, 8, 13, 10, 11, 2, 6))$
-     CP$(1, 8, (5, 7, 1, 3, 4, 8, 2, 6))$
-         CP$(1, 4, (1, 3, 4, 2))$
-         CP$(5, 8, (5, 7, 8, 6))$
-     CP$(9, 16, (16, 9, 15, 14, 12, 13, 10, 11))$

Outline

1. Divide-and-Conquer
2. Counting Inversions
3. Solving Recurrences
4. Quicksort and Selection
   - Quicksort
   - Lower Bound for Comparison-Based Sorting Algorithms
   - Selection Problem
5. Polynomial Multiplication
6. Strassen's Algorithm for Matrix Multiplication
7. FFT(Fast Fourier Transform): Polynomial Multiplication in $O(n \log n)$ Time
8. Finding Closest Pair of Points in 2D Euclidean Space
9. Computing n-th Fibonacci Number

Fibonacci Numbers

- $F_0 = 0, F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}, \forall n \geq 2$
- Fibonacci sequence: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \cdots$

n-th Fibonacci Number

Input: integer $n > 0$

Output: F_n

Computing F_n : Stupid Divide-and-Conquer Algorithm

Fib(n)

```
if $n = 0$ return 0
if $n = 1$ return 1
return Fib($n - 1$) + Fib($n - 2$)
```

Q: Is the running time of the algorithm polynomial or exponential in n?

Computing F_n : Stupid Divide-and-Conquer Algorithm

Fib(n)

```
if $n = 0$ return 0
if $n = 1$ return 1
return Fib$(n-1)$ + Fib$(n-2)$
```

Q: Is the running time of the algorithm polynomial or exponential in n?

A: Exponential

Computing F_n : Stupid Divide-and-Conquer Algorithm

Fib(n)

```
if $n = 0$ return 0
if $n = 1$ return 1
return Fib$(n-1)$ + Fib$(n-2)$
```

Q: Is the running time of the algorithm polynomial or exponential in n?

A: Exponential

- Running time is at least $\Omega(F_n)$

Computing F_n : Stupid Divide-and-Conquer Algorithm

Fib(n)

```
if $n = 0$ return 0
if $n = 1$ return 1
return Fib($n-1$) + Fib($n-2$)
```

Q: Is the running time of the algorithm polynomial or exponential in n?

A: Exponential

- Running time is at least $\Omega(F_n)$
- F_n is exponential in n

Computing F_n: Reasonable Algorithm

Fib(n)

```
$F[0] \leftarrow 0$
$F[1] \leftarrow 1$
for $i \leftarrow 2$ to $n$ do
    $F[i] \leftarrow F[i-1] + F[i-2]$
return $F[n]$
```

- Dynamic Programming

Computing F_n: Reasonable Algorithm

Fib(n)

```
$F[0] \leftarrow 0$
$F[1] \leftarrow 1$
for $i \leftarrow 2$ to $n$ do
    $F[i] \leftarrow F[i-1] + F[i-2]$
return $F[n]$
```

- Dynamic Programming
- Running time $=$?

Computing F_n: Reasonable Algorithm

Fib(n)

```
$F[0] \leftarrow 0$
$F[1] \leftarrow 1$
**for** $i \leftarrow 2$ to $n$ **do**
    $F[i] \leftarrow F[i-1] + F[i-2]$
**return** $F[n]$
```

- Dynamic Programming
- Running time $= O(n)$

Computing F_n: Even Better Algorithm

$$
\begin{aligned}
\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix} \\
\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2 \begin{pmatrix} F_{n-2} \\ F_{n-3} \end{pmatrix} \\
& \cdots \\
\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}
\end{aligned}
$$

power(n)

```
if $n = 0$ then return $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
$R \leftarrow$ power($\lfloor n/2 \rfloor$)
$R \leftarrow R \times R$
if $n$ is odd then $R \leftarrow R \times \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
**return** $R$
```

Fib(n)

```
if $n = 0$ then return $0$
$M \leftarrow$ power($n - 1$)
**return** $M[1][1]$
```

power(n)

```
if $n = 0$ then return $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
$R \leftarrow$ power($\lfloor n/2 \rfloor$)
$R \leftarrow R \times R$
if $n$ is odd then $R \leftarrow R \times \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
**return** $R$
```

Fib(n)

```
if $n = 0$ then return $0$
$M \leftarrow$ power($n - 1$)
**return** $M[1][1]$
```

- Recurrence for running time?

power(n)

```
if $n = 0$ then return $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
$R \leftarrow$ power($\lfloor n/2 \rfloor$)
$R \leftarrow R \times R$
if $n$ is odd then $R \leftarrow R \times \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
**return** $R$
```

Fib(n)

```
if $n = 0$ then return $0$
$M \leftarrow$ power($n - 1$)
**return** $M[1][1]$
```

- Recurrence for running time? $T(n) = T(n/2) + O(1)$

power(n)

```
if $n = 0$ then return $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
$R \leftarrow$ power($\lfloor n/2 \rfloor$)
$R \leftarrow R \times R$
if $n$ is odd then $R \leftarrow R \times \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
**return** $R$
```

Fib(n)

```
if $n = 0$ then return $0$
$M \leftarrow$ power($n - 1$)
**return** $M[1][1]$
```

- Recurrence for running time? $T(n) = T(n/2) + O(1)$
- $T(n) = O(\log n)$

Running time $= O(\log n)$: We Cheated!

Running time $= O(\log n)$: We Cheated!

Q: How many bits do we need to represent $F(n)$?

Running time $= O(\log n)$: We Cheated!

Q: How many bits do we need to represent $F(n)$?

A: $\Theta(n)$

Running time = $O(\log n)$: We Cheated!

Q: How many bits do we need to represent $F(n)$?

A: $\Theta(n)$

- We can not add (or multiply) two integers of $\Theta(n)$ bits in $O(1)$ time

Running time $= O(\log n)$: We Cheated!

Q: How many bits do we need to represent $F(n)$?

A: $\Theta(n)$

- We can not add (or multiply) two integers of $\Theta(n)$ bits in $O(1)$ time
- Even printing $F(n)$ requires time much larger than $O(\log n)$

Running time $= O(\log n)$: We Cheated!

Q: How many bits do we need to represent $F(n)$?

A: $\Theta(n)$

- We can not add (or multiply) two integers of $\Theta(n)$ bits in $O(1)$ time
- Even printing $F(n)$ requires time much larger than $O(\log n)$

Fixing the Problem

To compute F_n, we need $O(\log n)$ basic arithmetic operations on integers

Summary: Divide-and-Conquer

- **Divide**: Divide instance into many smaller instances
- **Conquer**: Solve each of smaller instances recursively and separately
- **Combine**: Combine solutions to small instances to obtain a solution for the original big instance

Summary: Divide-and-Conquer

- **Divide**: Divide instance into many smaller instances
- **Conquer**: Solve each of smaller instances recursively and separately
- **Combine**: Combine solutions to small instances to obtain a solution for the original big instance
- Write down recurrence for running time
- Solve recurrence using master theorem

Summary: Divide-and-Conquer

- Merge sort, quicksort, count-inversions, closest pair, FFT, $\cdots$:
  $T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$

Summary: Divide-and-Conquer

- Merge sort, quicksort, count-inversions, closest pair, FFT, $\cdots$:
  $T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$
- Polynomial Multiplication:
  $T(n) = 3T(n/2) + O(n) \Rightarrow T(n) = O(n^{\log_2 3})$

Summary: Divide-and-Conquer

- Merge sort, quicksort, count-inversions, closest pair, FFT, $\cdots$:
  $T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$
- Polynomial Multiplication:
  $T(n) = 3T(n/2) + O(n) \Rightarrow T(n) = O(n^{\log_2 3})$
- Matrix Multiplication:
  $T(n) = 7T(n/2) + O(n^2) \Rightarrow T(n) = O(n^{\log_2 7})$

Summary: Divide-and-Conquer

- Merge sort, quicksort, count-inversions, closest pair, FFT, $\cdots$:
  $T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$
- Polynomial Multiplication:
  $T(n) = 3T(n/2) + O(n) \Rightarrow T(n) = O(n^{\log_2 3})$
- Matrix Multiplication:
  $T(n) = 7T(n/2) + O(n^2) \Rightarrow T(n) = O(n^{\log_2 7})$

- To improve running time, design better algorithm for "combine" step, or reduce number of recursions, ...