Divide-and-Conquer

授课老师: 栗师
南京大学计算机科学与技术系
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($A, n$)

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

Counting Inversions

Input: an sequence $A$ of $n$ numbers
Output: number of inversions in $A$

Example:

$10 \ 8 \ 15 \ 9 \ 12$

4 inversions (for convenience, using numbers, not indices):

$(10, 8), (10, 9), (15, 9), (15, 12)$
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 $A$
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 |
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:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>8</td>
<td>15</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>
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
```

4 inversions (for convenience, using numbers, not indices):

- $(10, 8)$
- $(10, 9)$
- $(15, 9)$
- $(15, 12)$
**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:**

- 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)

1: $c \leftarrow 0$
2: for every $i \leftarrow 1$ to $n - 1$ do
3:     for every $j \leftarrow i + 1$ to $n$ do
4:         if $A[i] > A[j]$ then $c \leftarrow c + 1$
5: return $c$
Divide-and-Conquer

A:  

\[
\begin{array}{c}
A: \\
\hline
B \\
\hline
C \\
\end{array}
\]

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

**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

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

**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

$C$: 5 7 9 25 29

$\text{total} = 0$
Counting Inversions between $B$ and $C$

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

$B$: \[3 \ 8 \ 12 \ 20 \ 32 \ 48\]

$C$: \[5 \ 7 \ 9 \ 25 \ 29\]

total = 0
Counting Inversions between $B$ and $C$

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

$B$: \[\begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
\end{array}\] \hspace{1cm} \text{total}= 0

$C$: \[\begin{array}{cccccc}
5 & 7 & 9 & 25 & 29 \\
\end{array}\] +0

3
Counting Inversions between $B$ and $C$

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

\[
\begin{array}{c}
B: & 3 & 8 & 12 & 20 & 32 & 48 \\
C: & 5 & 7 & 9 & 25 & 29 \\
\end{array}
\]

\[\text{total} = 0\]
Counting Inversions between $B$ and $C$

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

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

$\text{total} = 0$

- $B$: [3 5]
- $C$: [8 7 12 20 32 48 5 7 9 25 29]

+0
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

$\text{total} = 0$

+0

3 5
Counting Inversions between $B$ and $C$

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

\[\begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
5 & 7 & 9 & 25 & 29 \\
\end{array}\]

\[\text{total} = 0\]
Counting Inversions between $B$ and $C$

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

$B$: \[\begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
\end{array}\]

$C$: \[\begin{array}{cccccc}
5 & 7 & 9 & 25 & 29 \\
\end{array}\]

$\text{total} = 0$

+0

3 5 7
Counting Inversions between \( B \) and \( C \)

Count pairs \( i, j \) such that \( B[i] > C[j] \):

\[
\begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
\end{array}
\]

\[
\begin{array}{cccccc}
5 & 7 & 9 & 25 & 29 \\
\end{array}
\]

\[\text{total} = 2\]

\[
\begin{array}{cccc}
3 & 5 & 7 & 8 \\
\end{array}
\]

\[+0 \quad +2\]
Counting Inversions between $B$ and $C$

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

\[ B: \begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
\end{array} \quad \text{total} = 2 \\

\begin{array}{cccccc}
5 & 7 & 9 & 25 & 29 \\
\end{array} \]

\[ +0 \quad +2 \]

\[ \begin{array}{cccc}
3 & 5 & 7 & 8 \\
\end{array} \]
Counting Inversions between $B$ and $C$

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

$B$: \begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
\end{array}$ \hspace{1cm} \text{total} = 2

$C$: \begin{array}{cccccc}
5 & 7 & 9 & 25 & 29 \\
\end{array}$

+0 +2

\begin{array}{cccccc}
3 & 5 & 7 & 8 & 9 \\
\end{array}$
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 = 2

+0 +2

3 5 7 8 9
Counting Inversions between $B$ and $C$

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

$B$: \[
\begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
\end{array}
\]

$C$: \[
\begin{array}{cccccc}
5 & 7 & 9 & 25 & 29 \\
\end{array}
\]

\[
\begin{array}{cccc}
+0 & +2 & +3 \\
3 & 5 & 7 & 8 & 9 & 12 \\
\end{array}
\]

Total = 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

$B$:

$C$:

$\text{total} = 5$

$\text{total} = 5$

$+0 +2 +3$

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  
C: 5 7 9 25 29  

+0 +2 +3 +3 

3 5 7 8 9 12 20  

$\text{total} = 8$
Counting Inversions between $B$ and $C$

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

$B$: \[ \begin{array}{cccccc} 3 & 8 & 12 & 20 & 32 & 48 \end{array} \]

$C$: \[ \begin{array}{cccccc} 5 & 7 & 9 & 25 & 29 \end{array} \]

Total = 8

The total number of inversions is 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  

3 +0 8 +2 12 +3 20 +3 48 25  

$\text{total} = 8$
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

$\text{total} = 8$

+0 +2 +3 +3

3 5 7 8 9 12 20 25
Counting Inversions between $B$ and $C$

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

- $B$: \[\begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
\end{array}\]
- $C$: \[\begin{array}{cccccc}
5 & 7 & 9 & 25 & 29 \\
\end{array}\]

Total: \[+0 +2 +3 +3\]

\[\begin{array}{cccccccccc}
3 & 5 & 7 & 8 & 9 & 12 & 20 & 25 & 29 \\
\end{array}\]
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

$B$: 3 8 12 20 32 48

$C$: 5 7 9 25 29

total = 8

+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 $B[i] > C[j]$:

$B: \begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
\end{array}$

$C: \begin{array}{cccccc}
5 & 7 & 9 & 25 & 29 & \\
\end{array}$

$\begin{array}{ccccccc}
+0 & +2 & +3 & +3 & +5 & \\
\end{array}$

$\begin{array}{cccccccccccc}
3 & 5 & 7 & 8 & 9 & 12 & 20 & 25 & 29 & 32 & \\
\end{array}$

$\text{total} = 13$
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 = 13
Counting Inversions between $B$ and $C$

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

$B$: [3, 8, 12, 20, 32, 48]  \quad \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 |  

$C$:  

| 5 | 7 | 9 | 25 | 29 |  

$\text{total} = 18$

\[ +0 \quad +2 \quad +3 \quad +3 \quad +5 \quad +5 \]

\[
| 3 | 5 | 7 | 8 | 9 | 12 | 20 | 25 | 29 | 32 | 48 |
\]
Count Inversions between $B$ and $C$

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

```plaintext
merge-and-count($B, C, n_1, n_2$)
1:  count ← 0;
2:  $A$ ← array of size $n_1 + n_2$; $i$ ← 1; $j$ ← 1
3:  while $i \leq n_1$ or $j \leq n_2$ do
4:    if $j > n_2$ or ($i \leq n_1$ and $B[i] \leq C[j]$) then
5:      $A[i + j - 1] ← B[i]$; $i ← i + 1$
6:    count ← count + ($j - 1$)
7:    else
8:      $A[i + j - 1] ← C[j]$; $j ← j + 1$
9:  return ($A, count$)
```
A procedure that returns the sorted array of $A$ and counts the number of inversions in $A$:

\begin{algorithm}
\begin{algorithmic}[1]
\Procedure{sort-and-count}{$A, n$}
\If{$n = 1$}
\Return $(A, 0)$
\Else
\State $(B, m_1) \leftarrow \text{sort-and-count}(A[1..\lfloor n/2 \rfloor], \lfloor n/2 \rfloor)$
\State $(C, m_2) \leftarrow \text{sort-and-count}(A[\lceil n/2 \rceil + 1..n], \lceil n/2 \rceil)$
\State $(A, m_3) \leftarrow \text{merge-and-count}(B, C, \lfloor n/2 \rfloor, \lceil n/2 \rceil)$
\Return $(A, m_1 + m_2 + m_3)$
\EndProcedure
\end{algorithmic}
\end{algorithm}
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($A[1..\lfloor n/2 \rfloor], \lfloor n/2 \rfloor$)
  5: $(C, m_2) \leftarrow$ sort-and-count($A[\lceil n/2 \rceil + 1..n], \lceil n/2 \rceil$)
  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\((A, n)\)

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

Recurrence for the running time: \(T(n) = 2T(n/2) + O(n)\)
sort-and-count\((A, n)\)

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

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

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

\[
(39, 8, 25, 47)
\]

\[
(8, 39) \quad 1
\]

\[
(25, 47)
\]

\[
(39, 8) \quad (25, 47)
\]
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)

sort-and-count(39, 8, 25, 47)

sort-and-count(39, 8)

sort-and-count(8, 39)

sort-and-count(8, 25, 39, 47)

(8, 25, 39, 47)

1 + 0 + 1 = 2
Example

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)

(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)

(8, 39)

(8, 39)

(8, 25, 39, 47)

\[1 + 0 + 1 = 2\]

(25, 47)

(25, 47)

(8, 25, 39, 47)

(12, 18, 41, 7)

(12, 18)

(12, 18)

(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)

(39, 8)

(8, 39)

1

(8, 25, 39, 47)

1 + 0 + 1 = 2

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

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

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

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

\[
(39, 8, 25, 47)
\]

\[
(8, 39) \rightarrow (8, 25, 39, 47) \rightarrow (8, 25, 39, 47)
\]

\[
1 + 0 + 1 = 2
\]

\[
(25, 47)
\]

\[
0
\]

\[
(8, 25, 39, 47)
\]

\[
(25, 47)
\]

\[
1
\]

\[
(12, 18, 41, 7)
\]

\[
(39, 8)
\]

\[
(25, 47)
\]

\[
(39, 8)
\]

\[
(41, 7)
\]

\[
(12, 18)
\]

\[
(7, 41)
\]

\[
(39, 8, 25, 47, 12, 18, 41, 7)
\]

\[
(39, 8, 25, 47)
\]

\[
(8, 39)
\]

\[
1
\]

\[
(5, 25, 39, 47)
\]

\[
(25, 47)
\]

\[
0
\]

\[
(39, 8, 25, 47)
\]

\[
(8, 39)
\]

\[
1
\]

\[
(25, 47)
\]

\[
0
\]

\[
(39, 8, 25, 47)
\]

\[
(8, 25, 39, 47)
\]

\[
1 + 0 + 1 = 2
\]
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)

sort-and-count(39, 8)

sort-and-count(39)

sort-and-count(39)

sort-and-count(8, 25, 39, 47)

sort-and-count(8, 39)

sort-and-count(8)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(39, 8, 25, 47)

(39, 8)

(25, 47)

(39, 8)

(41, 7)
Example

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

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

\[
\text{sort-and-count}(39, 8, 25, 47)
\]

\[
\text{sort-and-count}(8, 25, 39, 47)
\]

\[
1 + 0 + 1 = 2
\]

\[
(7, 8, 12, 18, 25, 39, 41, 47)
\]

\[
2 + 3 + 11 = 16
\]

\[
(6, 9, 14, 20, 29, 30, 33, 42)
\]

\[
17
\]

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

\[
(6, 9, 14, 20, 29, 30, 33, 42)
\]

\[
17
\]
Example

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

\[ 2 + 3 + 11 = 16 \]

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

\[ 16 + 17 + 33 = 66 \]

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

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

\[ 17 \]

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

\[ 16 + 17 + 33 = 66 \]
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

\[ T(n) = 3T(n/2) + O(n) \]
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 Diagram]
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

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

![Recursion Tree]

- 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

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

![Recursion Tree Diagram]

- 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

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

![Recursion Tree Diagram]

- 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 Diagram]

Index of last level: \( \log_2 n \)

Total running time: \( \log_2 n \times \sum_{i=0}^{\log_2 n} 3^i n^2 = O(n^2) \)
Recursion-Tree Method

- \( T(n) = 3T(n/2) + O(n^2) \)
Recursion-Tree Method

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

![Recursion Tree](image)
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 $i$: $(\frac{n}{2^i})^2 \times 3^i = (\frac{3}{4})^i n^2$

- Index of last level?
Recursion-Tree Method

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

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

- Index of last level: $\log_2 n$
Recursion-Tree Method

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

![Recursion Tree Diagram]

- 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 \( i \)? \((\frac{n}{2^i})^2 \times 3^i = \left(\frac{3}{4}\right)^i n^2\)

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

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

![Recursion Tree Diagram]

- 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

<table>
<thead>
<tr>
<th>Recurrences</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) = 2T(n/2) + O(n) )</td>
<td></td>
<td></td>
<td></td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>( T(n) = 3T(n/2) + O(n) )</td>
<td></td>
<td></td>
<td></td>
<td>( O(n^{\log_2 3}) )</td>
</tr>
<tr>
<td>( T(n) = 3T(n/2) + O(n^2) )</td>
<td></td>
<td></td>
<td></td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Recurrences</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n) = 2T(n/2) + O(n)$</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n)$</td>
<td></td>
<td></td>
<td></td>
<td>$O(n^{\log_2 3})$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n^2)$</td>
<td></td>
<td></td>
<td></td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Recurrences</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) = 2T(n/2) + O(n) )</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>( T(n) = 3T(n/2) + O(n) )</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>( O(n^{\log_2 3}) )</td>
</tr>
<tr>
<td>( T(n) = 3T(n/2) + O(n^2) )</td>
<td></td>
<td></td>
<td></td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Recurrences</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n) = 2T(n/2) + O(n)$</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n)$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>$O(n^{\log_2 3})$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n^2)$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

**Theorem**  

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

<table>
<thead>
<tr>
<th>Recurrences</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) = 2T(n/2) + O(n) )</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>( T(n) = 3T(n/2) + O(n) )</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>( O(n^{\log_2 3}) )</td>
</tr>
<tr>
<td>( T(n) = 3T(n/2) + O(n^2) )</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Recurrences</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n) = 2T(n/2) + O(n)$</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n)$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>$O(n^{\log_2 3})$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n^2)$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Recurrences</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) = 2T(n/2) + O(n) )</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>( T(n) = 3T(n/2) + O(n) )</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>( O(n^{\log_2 3}) )</td>
</tr>
<tr>
<td>( T(n) = 3T(n/2) + O(n^2) )</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Recurrences</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n) = 2T(n/2) + O(n)$</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n)$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>$O(n^{\log_2 3})$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n^2)$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Recurrences</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n) = 2T(n/2) + O(n)$</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n)$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>$O(n^{\log_2 3})$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n^2)$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

Theorem

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

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

<table>
<thead>
<tr>
<th>Recurrences</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n) = 2T(n/2) + O(n)$</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n)$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>$O(n^{\log_2 3})$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n^2)$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Recurrences</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n) = 2T(n/2) + O(n)$</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n)$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>$O(n^{\log_2 3})$</td>
</tr>
<tr>
<td>$T(n) = 3T(n/2) + O(n^2)$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

**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

\[ n^c \]

\[ (n/b)^c \]

\[ a \text{ nodes} \]

\[ (n/b)^c \]

\[ (n/b)^c \]

\[ a^2 \text{ nodes} \]

\[ (n/b^2)^c \]

\[ (n/b^2)^c \]

\[ (n/b^2)^c \]

\[ a^3 \text{ nodes} \]

\[ (n/b^3)^c \]

\[ (n/b^3)^c \]

\[ (n/b^3)^c \]

. . . . . . . . . .
Proof of Master Theorem Using Recursion Tree

\[ T(n) = aT\left( \frac{n}{b} \right) + O(n^c) \]

1 node

\[ n^c \]

\[ \frac{n^c}{a} \]

\[ (n/b)^c \]

\[ (n/b)^c \]

\[ a \text{ nodes} \]

\[ \frac{n^c}{b^3} \]

\[ \frac{n^c}{b^3} \]

\[ (n/b^2)^c \]

\[ (n/b^2)^c \]

\[ a^2 \text{ nodes} \]

\[ \frac{n^c}{b^9} \]

\[ \frac{n^c}{b^9} \]

\[ (n/b^3)^c \]

\[ (n/b^3)^c \]

\[ (n/b^3)^c \]

\[ (n/b^3)^c \]

\[ a^3 \text{ nodes} \]

\[ \frac{n^c}{b^{27}} \]

\[ \frac{n^c}{b^{27}} \]

\[ \frac{n^c}{b^{27}} \]

\[ \frac{n^c}{b^{27}} \]

\[ \frac{n^c}{b^{27}} \]

\[ \frac{n^c}{b^{27}} \]

\[ \frac{n^c}{b^{27}} \]

\[ \frac{n^c}{b^{27}} \]

\[ \frac{n^c}{b^{27}} \]
Proof of Master Theorem Using Recursion Tree

\[
T(n) = aT(n/b) + O(n^c)
\]

1 node

\( n^c \)

\( n^c \)

a nodes

\( (n/b)^c \)

\( (n/b)^c \)

\( \frac{a}{b} n^c \)

\( a^2 \) 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

\( \left(\frac{n}{b^3}\right)^c \)

\( \left(\frac{n}{b^3}\right)^c \)

\( \left(\frac{n}{b^3}\right)^c \)

\( \left(\frac{n}{b^3}\right)^c \)

\( \left(\frac{n}{b^3}\right)^c \)

\( \left(\frac{n}{b^3}\right)^c \)

\( \left(\frac{n}{b^3}\right)^c \)

\( \left(\frac{a}{b^c}\right)^3 n^c \)

- \( c < \log_b a \) : 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

\[ T(n) = aT(n/b) + O(n^c) \]

![Recursion Tree Diagram]

- \( c < \log_b a \): bottom-level dominates: \((\frac{a}{b^c})^{\log_b n} n^c = n^{\log_b a}\)
- \( c = \log_b a \): all levels have same time: \( n^c \log_b n = O(n^c \log n)\)
Proof of Master Theorem Using Recursion Tree

\[ T(n) = aT\left(\frac{n}{b}\right) + O(n^c) \]

- 1 node
- \(a\) nodes
- \(a^2\) nodes
- \(a^3\) nodes

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

<table>
<thead>
<tr>
<th>Divide</th>
<th>Merge Sort</th>
<th>QuickSort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conquer</td>
<td>Trivial</td>
<td>Separate small and big numbers</td>
</tr>
<tr>
<td>Combine</td>
<td>Merge 2 sorted arrays</td>
<td>Recurse</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trivial</td>
</tr>
</tbody>
</table>
Quicksort Example

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

$$A: 
\begin{array}{cccccccccccccc}
29 & 82 & 75 & 64 & 38 & 45 & 94 & 69 & 25 & 76 & 15 & 92 & 37 & 17 & 85 \\
\end{array}$$

quicksort($A, 1, 15$)
Quicksort Example

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

\[
\begin{array}{cccccccccccc}
A: & 29 & 82 & 75 & 64 & 38 & 45 & 94 & 69 & 25 & 76 & 15 & 92 & 37 & 17 & 85 \\
\end{array}
\]

\[
\text{quicksort}(A, 1, 15)
\]
Quicksort Example

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

\[
A:
\begin{array}{cccccccccccccc}
29 & 38 & 45 & 25 & 15 & 37 & 17 & \mathbf{64} & 82 & 75 & 94 & 92 & 69 & 76 & 85 \\
\end{array}
\]

\[ \text{quicksort}(A, 1, 15) \]
Quicksort Example

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

$$A: \begin{array}{cccccccccccccc}
29 & 38 & 45 & 25 & 15 & 37 & 17 & 64 & 82 & 75 & 94 & 92 & 69 & 76 & 85
\end{array}$$

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

$$\text{quicksort}(A, 1, 7)$$
**Assumption**  We can choose median of an array of size $n$ in $O(n)$ time.

\[
A:
\begin{array}{cccccccccccc}
29 & 38 & 45 & 25 & 15 & 37 & 17 & 64 & 82 & 75 & 94 & 92 & 69 & 76 & 85
\end{array}
\]

\[\text{quicksort}(A, 1, 15)\]

\[\text{quicksort}(A, 1, 7)\]
Quicksort Example

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

\[
A:
\begin{array}{ccccccccccccc}
25 & 15 & 17 & 29 & 38 & 45 & 37 & 64 & 82 & 75 & 94 & 92 & 69 & 76 & 85 \\
\end{array}
\]

\[
\text{quicksort}(A, 1, 15)
\]

\[
\text{quicksort}(A, 1, 7)
\]
**Quicksort Example**

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

$$A:$$

<table>
<thead>
<tr>
<th>25</th>
<th>15</th>
<th>17</th>
<th>29</th>
<th>38</th>
<th>45</th>
<th>37</th>
<th>64</th>
<th>82</th>
<th>75</th>
<th>94</th>
<th>92</th>
<th>69</th>
<th>76</th>
<th>85</th>
</tr>
</thead>
</table>

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

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

$$\text{quicksort}(A, 1, 3)$$
**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)
Assumption  We can choose median of an array of size $n$ in $O(n)$ time.

$A$:  

\begin{array}{cccccccccccc}
15 & 17 & 25 & 29 & 38 & 45 & 37 & 64 & 82 & 75 & 94 & 92 & 69 & 76 & 85
\end{array}

\begin{align*}
\text{quicksort}(A, 1, 15) \\
&\text{quicksort}(A, 1, 7) \\
&\text{quicksort}(A, 1, 3) \quad \text{quicksort}(A, 5, 7)
\end{align*}
Quicksort Example

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

\[
A: \begin{bmatrix}
15 & 17 & 25 & 29 & 37 & 38 & 45 & 64 & 82 & 75 & 94 & 92 & 69 & 76 & 85
\end{bmatrix}
\]

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($A$, 9, 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 | 82 | 75 | 94 | 92 | 69 | 76 | 85 |

- **quicksort**($A, 1, 15$)  
  - **quicksort**($A, 1, 7$)  
    - **quicksort**($A, 1, 3$)  
    - **quicksort**($A, 5, 7$)  
  - **quicksort**($A, 9, 15$)
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 | 75 | 69 | 76 | 82 | 92 | 94 | 85 |

```
quicksort(A, 1, 15)
```

```
quicksort(A, 1, 7)
```

```
quicksort(A, 1, 3)
```

```
quicksort(A, 9, 15)
```

```
quicksort(A, 5, 7)
```
**Assumption**  We can choose median of an array of size $n$ in $O(n)$ time.

$$A: \begin{array}{cccccccccccc}
15 & 17 & 25 & 29 & 37 & 38 & 45 & 64 & 75 & 69 & 76 & 82 & 92 & 94 & 85 \\
\end{array}$$

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

$$\begin{array}{c}
\text{quicksort}(A, 1, 7) \\
\text{quicksort}(A, 5, 7) \\
\text{quicksort}(A, 9, 11) \\
\end{array}$$

$$\begin{array}{c}
\text{quicksort}(A, 1, 3) \\
\text{quicksort}(A, 9, 15) \\
\end{array}$$
Assumption  We can choose median of an array of size $n$ in $O(n)$ time.

$A$:

\[
\begin{array}{cccccccccccc}
15 & 17 & 25 & 29 & 37 & 38 & 45 & 64 & 69 & 75 & 76 & 82 & 92 & 94 & 85 \\
\end{array}
\]
**Quicksort Example**

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

\[
A:
\begin{array}{cccccccccccc}
15 & 17 & 25 & 29 & 37 & 38 & 45 & 64 & 69 & 75 & 76 & 82 & 92 & 94 & 85 \\
\end{array}
\]

**Diagram:**

```
Quicksort(A, 1, 15)

  Quicksort(A, 1, 7)
      Quicksort(A, 1, 3)
      Quicksort(A, 5, 7)
  Quicksort(A, 9, 15)
      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($A$, 1, 3)
  - Quicksort($A$, 5, 7)

- Quicksort($A$, 9, 15)
  - Quicksort($A$, 9, 11)
  - Quicksort($A$, 13, 15)
Quicksort

quicksort\((A, n)\)

1: if \( n \leq 1 \) then return \( A \)
2: \( x \leftarrow \) lower median of \( A \)
3: \( A_L \leftarrow \) array of elements in \( A \) that are less than \( x \)
4: \( A_R \leftarrow \) array of elements in \( A \) that are greater than \( x \)
5: \( B_L \leftarrow \) quicksort\((A_L, \text{length of } A_L)\)
6: \( B_R \leftarrow \) quicksort\((A_R, \text{length of } A_R)\)
7: \( t \leftarrow \) number of times \( x \) appear in \( A \)
8: return concatenation of \( B_L \), \( t \) copies of \( x \), and \( B_R \)
Quicksort

```plaintext
quicksort(A, n)
1: if n ≤ 1 then return A
2: x ← lower median of A
3: AL ← array of elements in A that are less than x \ Divide
4: AR ← array of elements in A that are greater than x \ Divide
5: BL ← quicksort(AL, length of AL) \ Conquer
6: BR ← quicksort(AR, length of AR) \ Conquer
7: t ← number of times x appear A
8: return concatenation of BL, t copies of x, and BR

Recurrence T(n) ≤ 2T(n/2) + O(n)
```
**Quicksort**

quicksort\((A, n)\)

1: if \( n \leq 1 \) then return \( A \)
2: \( x \leftarrow \) lower median of \( A \)
3: \( A_L \leftarrow \) array of elements in \( A \) that are less than \( x \) \quad \| \quad \text{Divide}
4: \( A_R \leftarrow \) array of elements in \( A \) that are greater than \( x \) \quad \| \quad \text{Divide}
5: \( B_L \leftarrow \) quicksort\((A_L, \text{length of } A_L)\) \quad \| \quad \text{Conquer}
6: \( B_R \leftarrow \) quicksort\((A_R, \text{length of } A_R)\) \quad \| \quad \text{Conquer}
7: \( t \leftarrow \) number of times \( x \) appear in \( A \)
8: return concatenation of \( B_L \), \( t \) copies of \( x \), and \( B_R \)

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

1: if \( n \leq 1 \) then return A
2: \( x \leftarrow \) a random element of A (\( x \) is called a pivot)
3: \( A_L \leftarrow \) array of elements in A that are less than \( x \) \( \text{Divide} \)
4: \( A_R \leftarrow \) array of elements in A that are greater than \( x \) \( \text{Divide} \)
5: \( B_L \leftarrow \) quicksort\((A_L, \text{length of } A_L)\) \( \text{Conquer} \)
6: \( B_R \leftarrow \) quicksort\((A_R, \text{length of } A_R)\) \( \text{Conquer} \)
7: \( t \leftarrow \) number of times \( x \) appear in A
8: 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 program is deterministic!
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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: if $n \leq 1$ then return $A$</td>
<td></td>
</tr>
<tr>
<td>2: $x \leftarrow$ a random element of $A$ ($x$ is called a pivot)</td>
<td></td>
</tr>
<tr>
<td>3: $A_L \leftarrow$ array of elements in $A$ that are less than $x$</td>
<td>Divide</td>
</tr>
<tr>
<td>4: $A_R \leftarrow$ array of elements in $A$ that are greater than $x$</td>
<td>Divide</td>
</tr>
<tr>
<td>5: $B_L \leftarrow$ quicksort($A_L$, length of $A_L$)</td>
<td>Conquer</td>
</tr>
<tr>
<td>6: $B_R \leftarrow$ quicksort($A_R$, length of $A_R$)</td>
<td>Conquer</td>
</tr>
<tr>
<td>7: $t \leftarrow$ number of times $x$ appear</td>
<td></td>
</tr>
<tr>
<td>8: return concatenation of $B_L$, $t$ copies of $x$, and $B_R$</td>
<td></td>
</tr>
</tbody>
</table>

**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.
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.

To partition the array into two parts, we only need $O(1)$ extra space.
Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

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

To partition the array into two parts, we only need $O(1)$ extra space.
Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

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

\[
\begin{array}{cccccccccccc}
17 & 82 & 75 & 29 & 38 & 45 & 94 & 69 & 25 & 76 & 15 & 92 & 37 & 64 & 85 \\
\end{array}
\]
Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

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

![Array partition](image)

To partition the array into two parts, we only need $O(1)$ extra space.
QuickSort Can Be Implemented as an “In-Place” Sorting Algorithm

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

To partition the array into two parts, we only need $O(1)$ extra space.
Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

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

To partition the array into two parts, we only need $O(1)$ extra space.
Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

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

\[
i < j
\]

\[
17 \ 37 \ 75 \ 29 \ 38 \ 45 \ 94 \ 69 \ 25 \ 76 \ 15 \ 92 \ 64 \ 82 \ 85
\]
Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

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

To partition the array into two parts, we only need $O(1)$ extra space.
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.
Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

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

To partition the array into two parts, we only need $O(1)$ extra space.
Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

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

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

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

To partition the array into two parts, we only need $O(1)$ extra space.
Quicksort Can Be Implemented as an “In-Place” Sorting Algorithm

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

![Array with indices i and j indicating partitioning]

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

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

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.

To partition the array into two parts, we only need $O(1)$ extra space.
QuickSort can be implemented as an “in-place” sorting algorithm.

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

To partition the array into two parts, we only need $O(1)$ extra space.
partition\((A, l, r)\)

1: \( p \leftarrow \text{random integer between } l \text{ and } r, \text{ swap } A[p] \text{ and } A[l] \)
2: \( i \leftarrow l, j \leftarrow r \)
3: \textbf{while true do}
4: \hspace{1em} \textbf{while } i < j \text{ and } A[i] < A[j] \textbf{ do } j \leftarrow j - 1
5: \hspace{1em} \textbf{if } i = j \textbf{ then break}
6: \hspace{1em} \text{swap } A[i] \text{ and } A[j]; \hspace{1em} i \leftarrow i + 1
7: \hspace{1em} \textbf{while } i < j \text{ and } A[i] < A[j] \textbf{ do } i \leftarrow i + 1
8: \hspace{1em} \textbf{if } i = j \textbf{ then break}
9: \hspace{1em} \text{swap } A[i] \text{ and } A[j]; \hspace{1em} j \leftarrow j - 1
10: \hspace{1em} \textbf{return } i
In-Place Implementation of Quick-Sort

quicksort(A, ℓ, r)

1: if ℓ ≥ r then return
2: m ← partition(A, ℓ, r)
3: quicksort(A, ℓ, m − 1)
4: 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

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>12</td>
<td>20</td>
<td>32</td>
<td>48</td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>7</td>
<td>9</td>
<td>25</td>
<td>29</td>
<td></td>
</tr>
</tbody>
</table>
Merge-Sort is Not In-Place

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

\[
\begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
5 & 7 & 9 & 25 & 29 \\
\end{array}
\]
Merge-Sort is Not In-Place

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

\[
\begin{array}{cccccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
5 & 7 & 9 & 25 & 29 \\
3
\end{array}
\]
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
Merge-Sort is Not In-Place

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

\[
\begin{array}{ccccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
5 & 7 & 9 & 25 & 29 \\
3 & 5 \\
\end{array}
\]
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.
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
```
To merge two arrays, we need a third array with size equaling the total size of two arrays.

\[
\begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
5 & 7 & 9 & 25 & 29 \\
3 & 5 & 7 & 8 & 9 & 12 & 20 & 25 & 29 & 32 & 48
\end{array}
\]
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
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 $x$. 

$\begin{align*}
&\text{Q: } x = 1? \\
&\text{A: } \lceil \log_2 N \rceil
\end{align*}$
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, $x \in \{1, 2, 3, \ldots, N\}$.
- 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:** $\lceil \log_2 N \rceil$.
Q: Can we do better than $O(n \log n)$ for sorting?

A: No, for comparison-based sorting algorithms.

- Bob has a permutation $\pi$ over $\{1, 2, 3, \ldots, n\}$ in his hand.
- You can ask Bob “yes/no” questions about $\pi$. 
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 $\pi$ over $\{1, 2, 3, \ldots, n\}$ in his hand.
- You can ask Bob “yes/no” questions about $\pi$.

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

$\log_2 n! = \Theta(n \log n)$
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 $\pi$ over $\{1, 2, 3, \ldots, n\}$ in his hand.
- You can ask Bob “yes/no” questions about $\pi$.

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

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 $\pi$ over $\{1, 2, 3, \ldots, n\}$ in his hand.
- You can ask Bob questions of the form “does $i$ appear before $j$ in $\pi$?”
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 $\pi$ over $\{1, 2, 3, \cdots, n\}$ in his hand.
- You can ask Bob questions of the form “does $i$ appear before $j$ in $\pi$?”

Q: How many questions do you need to ask in order to get the permutation $\pi$?
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 $\pi$ over $\{1, 2, 3, \cdots, n\}$ in his hand.
- You can ask Bob questions of the form “does $i$ appear before $j$ in $\pi$?”

**Q:** How many questions do you need to ask in order to get the permutation $\pi$?

**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 $1 \leq i \leq n$

**Output:** the $i$-th smallest number in $A$
Selection Problem

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

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 $1 \leq i \leq n$

**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

\begin{algorithmic}
  \State \textbf{if} $n \leq 1$ \textbf{then return} $A$
  \State $x \leftarrow$ lower median of $A$
  \State $A_L \leftarrow$ elements in $A$ that are less than $x$ \hspace{1cm} \triangleright \text{Divide}
  \State $A_R \leftarrow$ elements in $A$ that are greater than $x$ \hspace{1cm} \triangleright \text{Divide}
  \State $B_L \leftarrow$ quicksort($A_L, A_L.\text{size}$) \hspace{1cm} \triangleright \text{Conquer}
  \State $B_R \leftarrow$ quicksort($A_R, A_R.\text{size}$) \hspace{1cm} \triangleright \text{Conquer}
  \State $t \leftarrow$ number of times $x$ appear $A$
  \State \textbf{return} the array obtained by concatenating $B_L$, the array containing $t$ copies of $x$, and $B_R$
\end{algorithmic}
Selection Algorithm with Median Finder

**selection**(*A, n, i*)

1: if *n* = 1 then return *A*
2: *x* ← lower median of *A*
3: *A_L* ← elements in *A* that are less than *x*  ▷ Divide
4: *A_R* ← elements in *A* that are greater than *x*  ▷ Divide
5: if *i* ≤ *A_L*.size then
6: return **selection**(*A_L, A_L.size, i*)  ▷ Conquer
7: else if *i* > *n* − *A_R*.size then
8: return **selection**(*A_R, A_R.size, i − (n − A_R.size)) ▷ Conquer
9: else
10: return *x*
Selection Algorithm with Median Finder

**selection**($A, n, i$)

1: if $n = 1$ then return $A$
2: $x \leftarrow$ lower median of $A$
3: $A_L \leftarrow$ elements in $A$ that are less than $x$ ▷ Divide
4: $A_R \leftarrow$ elements in $A$ that are greater than $x$ ▷ Divide
5: if $i \leq A_L$.size then
6: return selection($A_L, A_L$.size, $i$) ▷ Conquer
7: else if $i > n - A_R$.size then
8: return selection($A_R, A_R$.size, $i - (n - A_R$.size)) ▷ Conquer
9: else
10: return $x$

Recurrence for selection: $T(n) = T(n/2) + O(n)$
Selection Algorithm with Median Finder

\[ \text{selection}(A, n, i) \]

1. \textbf{if} \( n = 1 \) \textbf{ then return } A
2. \( x \leftarrow \) lower median of \( A \)
3. \( A_L \leftarrow \) elements in \( A \) that are less than \( x \) \hspace{1cm} \text{ Divide}
4. \( A_R \leftarrow \) elements in \( A \) that are greater than \( x \) \hspace{1cm} \text{ Divide}
5. \textbf{if} \( i \leq A_L.\text{size} \) \textbf{ then}
6. \textbf{return} \( \text{selection}(A_L, A_L.\text{size}, i) \) \hspace{1cm} \text{ Conquer}
7. \textbf{else if} \( i > n - A_R.\text{size} \) \textbf{ then}
8. \textbf{return} \( \text{selection}(A_R, A_R.\text{size}, i - (n - A_R.\text{size})) \) \hspace{1cm} \text{ Conquer}
9. \textbf{else}
10. \textbf{return} \( x \)

- Recurrence for selection: \( T(n) = T(n/2) + O(n) \)
- Solving recurrence: \( T(n) = O(n) \)
### Randomized Selection Algorithm

**selection**\((A, n, i)\)

1. \textbf{if} \(n = 1\) \textbf{then return} \(A\)
2. \(x \leftarrow \text{random element of } A\) (called \textit{pivot})
3. \(A_L \leftarrow \text{elements in } A \text{ that are less than } x\) \hspace{1cm} \(\triangleright \text{Divide}\)
4. \(A_R \leftarrow \text{elements in } A \text{ that are greater than } x\) \hspace{1cm} \(\triangleright \text{Divide}\)
5. \textbf{if} \(i \leq A_L\text{.size}\) \textbf{then}
6. \hspace{1cm} \textbf{return} selection\((A_L, A_L\text{.size}, i)\) \hspace{1cm} \(\triangleright \text{Conquer}\)
7. \textbf{else if} \(i > n - A_R\text{.size}\) \textbf{then}
8. \hspace{1cm} \textbf{return} selection\((A_R, A_R\text{.size}, i - (n - A_R\text{.size}))\) \hspace{1cm} \(\triangleright \text{Conquer}\)
9. \textbf{else}
10. \hspace{1cm} \textbf{return} \(x\)
Randomized Selection Algorithm

**selection**\((A, n, i)\)

1: if \(n = 1\) then return \(A\)  
2: \(x \leftarrow\) random element of \(A\) (called \text{pivot})  
3: \(A_L \leftarrow\) elements in \(A\) that are less than \(x\) \ ▷ Divide  
4: \(A_R \leftarrow\) elements in \(A\) that are greater than \(x\) \ ▷ Divide  
5: if \(i \leq A_L\).size then  
6: return \(\text{selection}(A_L, A_L\text{.size}, i)\) \ ▷ Conquer  
7: else if \(i > n – A_R\).size then  
8: return \(\text{selection}(A_R, A_R\text{.size}, i – (n – A_R\text{.size}))\) \ ▷ Conquer  
9: else  
10: 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

Example:

$$(3x^3 + 2x^2 - 5x + 4) \times (2x^3 - 3x^2 + 6x - 5) = 6x^6 - 9x^5 + 18x^4 - 15x^3 + 4x^5 - 6x^4 + 12x^3 - 10x^2 - 10x^4 + 15x^3 - 30x^2 + 25x + 8x^3 - 12x^2 + 24x - 20$$
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:

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

$$= 6x^6 - 9x^5 + 18x^4 - 15x^3$$

$$+ 4x^5 - 6x^4 + 12x^3 - 10x^2$$

$$- 10x^4 + 15x^3 - 30x^2 + 25x$$

$$+ 8x^3 - 12x^2 + 24x - 20$$

$$= 6x^6 - 5x^5 + 2x^4 + 20x^3 - 52x^2 + 49x - 20$$
Polynomial Multiplication

**Input:** two polynomials of degree $n - 1$

**Output:** product of two polynomials

Example:

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

\[= 6x^6 - 9x^5 + 18x^4 - 15x^3 + 4x^5 - 6x^4 + 12x^3 - 10x^2 - 10x^4 + 15x^3 - 30x^2 + 25x + 8x^3 - 12x^2 + 24x - 20\]

\[= 6x^6 - 5x^5 + 2x^4 + 20x^3 - 52x^2 + 49x - 20\]

- **Input:** $(4, -5, 2, 3), (-5, 6, -3, 2)$
- **Output:** $(-20, 49, -52, 20, 2, -5, 6)$
Discrete Convolution on Finite Domain

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

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

Applications of Convolutions

- Polynomial and integer multiplication
- Signal and Image Processing
- Probability theory: Sum of two distributions
- Convolutional neural network
### Discrete Convolution on Finite Domain

- \( f : \{0, 1, \cdots, n - 1\} \rightarrow \mathbb{R}, \ g : \{0, 1, \cdots, m - 1\} \rightarrow \mathbb{R} \)

- The convolution of \( f \) and \( g \), denoted as \( h := f \times g \), is defined as

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>4</td>
<td>-5</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( g )</td>
<td>-5</td>
<td>6</td>
<td>-3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( f \times g )</td>
<td>-20</td>
<td>49</td>
<td>-52</td>
<td>20</td>
<td>2</td>
<td>-5</td>
<td>6</td>
</tr>
</tbody>
</table>
Discrete Convolution on Finite Domain

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

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>4</td>
<td>-5</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$g$</td>
<td>-5</td>
<td>6</td>
<td>-3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$f \times g$:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20</td>
<td>49</td>
<td>-52</td>
<td>20</td>
<td>2</td>
<td>-5</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

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 \iff 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 $\Leftrightarrow$ Convolution

We shall focus on multiplication.

**Big Integer Multiplication Using Polynomial Multiplication**

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

We shall focus on multiplication.

**Big Integer Multiplication Using Polynomial Multiplication**

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

**polynomial-multiplication***(\(A, B, n\))*

1: let \(C[k] \leftarrow 0\) for every \(k = 0, 1, 2, \ldots, 2n - 2\)
2: for \(i \leftarrow 0\) to \(n - 1\) do
3:     for \(j \leftarrow 0\) to \(n - 1\) do
4:         \(C[i + j] \leftarrow C[i + j] + A[i] \times B[j]\)
5: return \(C\)

Running time: \(O(n^2)\)
Naïve Algorithm

**polynomial-multiplication**\((A, B, n)\)

1: let \(C[k] \leftarrow 0\) for every \(k = 0, 1, 2, \ldots, 2n - 2\)
2: for \(i \leftarrow 0\) to \(n - 1\) do
3:     for \(j \leftarrow 0\) to \(n - 1\) do
4:         \(C[i + j] \leftarrow C[i + j] + A[i] \times B[j]\)
5: 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 \( n/2 - 1 \).

\[ pq = (p_Hx^{n/2} + p_L)(q_Hx^{n/2} + q_L) \]
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 \( n/2 - 1 \).

\[ pq = (p_H x^{n/2} + p_L)(q_H x^{n/2} + q_L) \]
\[ = p_H q_H x^n + (p_H q_L + p_L q_H) x^{n/2} + p_L q_L \]
Divide-and-Conquer for Polynomial Multiplication

\[ pq = (p_H x^{n/2} + p_L)(q_H x^{n/2} + q_L) = p_H q_H x^n + (p_H q_L + p_L q_H) x^{n/2} + p_L q_L \]
Divide-and-Conquer for Polynomial Multiplication

\[ pq = (p_H x^{n/2} + p_L) (q_H x^{n/2} + q_L) = p_H q_H x^n + (p_H q_L + p_L q_H) x^{n/2} + p_L q_L \]

\[
\text{multiply}(p, q) = \text{multiply}(p_H, q_H) \times x^n \\
+ (\text{multiply}(p_H, q_L) + \text{multiply}(p_L, q_H)) \times x^{n/2} \\
+ \text{multiply}(p_L, q_L)
\]
Divide-and-Conquer for Polynomial Multiplication

\[ pq = (p_H x^{n/2} + p_L)(q_H x^{n/2} + q_L) \]
\[ = p_H q_H x^n + (p_H q_L + p_L q_H) x^{n/2} + p_L q_L \]

\[
multiply(p, q) = multiply(p_H, q_H) \times x^n \]
\[ + \left( multiply(p_H, q_L) + multiply(p_L, q_H) \right) \times x^{n/2} \]
\[ + multiply(p_L, q_L) \]

- Recurrence: \( T(n) = 4T(n/2) + O(n) \)
Divide-and-Conquer for Polynomial Multiplication

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

\[
multiply(p, q) = multiply(p_H, q_H) \times x^n
+ \left( multiply(p_H, q_L) + multiply(p_L, q_H) \right) \times x^{n/2}
+ multiply(p_L, q_L)
\]

- 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

\[ pq = (p_H x^{n/2} + p_L) (q_H x^{n/2} + q_L) \]
\[ = p_H q_H x^n + (p_H q_L + p_L q_H) x^{n/2} + p_L q_L \]
Reduce Number from 4 to 3

\[ pq = (p_H x^{n/2} + p_L) (q_H x^{n/2} + q_L) \]
\[ = p_H q_H x^n + (p_H q_L + p_L q_H) x^{n/2} + p_L q_L \]

\[ p_H q_L + p_L q_H = (p_H + p_L) (q_H + q_L) - p_H q_H - p_L q_L \]
Divide-and-Conquer for Polynomial Multiplication

\[
H = \text{multiply}(p_H, q_H)
\]

\[
L = \text{multiply}(p_L, q_L)
\]

\[
multiply(p, q) = H \times x^n + \text{multiply}(p_H + p_L, q_H + q_L) - H - L \times x^{n/2} + L
\]

Solving Recurrence:

\[
T(n) = 3T(n/2) + O(n)
\]

\[
T(n) = O(n \log_2^3) = O(n^{1.585})
\]
Divide-and-Conquer for Polynomial Multiplication

\[ r_H = \text{multiply}(p_H, q_H) \]
\[ r_L = \text{multiply}(p_L, q_L) \]
Divide-and-Conquer for Polynomial Multiplication

\[ r_H = \text{multiply}(p_H, q_H) \]
\[ r_L = \text{multiply}(p_L, q_L) \]

\[
\text{multiply}(p, q) = r_H \times x^n \\
+ \left( \text{multiply}(p_H + p_L, q_H + q_L) - r_H - r_L \right) \times x^{n/2} \\
+ r_L
\]
Divide-and-Conquer for Polynomial Multiplication

\[ r_H = \text{multiply}(p_H, q_H) \]
\[ r_L = \text{multiply}(p_L, q_L) \]
\[ \text{multiply}(p, q) = r_H \times x^n \]
\[ + \left( \text{multiply}(p_H + p_L, q_H + q_L) - r_H - r_L \right) \times x^{n/2} \]
\[ + r_L \]

- Solving Recurrence: \( T(n) = 3T(n/2) + O(n) \)
Divide-and-Conquer for Polynomial Multiplication

\[ r_H = \text{multiply}(p_H, q_H) \]
\[ r_L = \text{multiply}(p_L, q_L) \]

\[
\text{multiply}(p, q) = r_H \times x^n \\
+ \left( \text{multiply}(p_H + p_L, q_H + q_L) - r_H - r_L \right) \times x^{n/2} \\
+ r_L
\]

- **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)
1: if n = 1 then return (A[0]B[0])
2: A_L ← A[0 .. n/2 − 1], A_H ← A[n/2 .. n − 1]
3: B_L ← B[0 .. n/2 − 1], B_H ← B[n/2 .. n − 1]
4: C_L ← multiply(A_L, B_L, n/2)
5: C_H ← multiply(A_H, B_H, n/2)
6: C_M ← multiply(A_L + A_H, B_L + B_H, n/2)
7: C ← array of (2n − 1) 0’s
8: for i ← 0 to n − 2 do
9: C[i] ← C[i] + C_L[i]
10: C[i + n] ← C[i + n] + C_H[i]
11: C[i + n/2] ← C[i + n/2] + C_M[i] − C_L[i] − C_H[i]
12: return C
```
Example

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

\[(3 + 2x + 2x^2 + 4x^3 + x^4 + 2x^5 + x^6 + 5x^7) \times (2 + x - x^2 + 2x^3 - 2x^4 - x^5 + 2x^6 - 2x^7)\]
Example

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

\[= (3 + 2x + 2x^2 + 4x^3) \times (2 + x - x^2 + 2x^3) \times (1 + 2x + x^2 + 5x^3) \times (-2 - x + 2x^2 - 2x^3) \times (4 + 4x + 3x^2 + 9x^3) \times x^2\]

\[= (3 + 2x) \times (2 + x) \times (2 + 4x) \times (-1 + 2x) \times (5 + 6x) \times (1 + 3x)\]

\[= (6 + 7x + 2x^2) \times (-2 + 8x^2) \times (5 + 21x + 18x^2)\]
Example

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

\[(3 + 2x + 2x^2 + 4x^3) \times (2 + x - x^2 + 2x^3)\]
\[(1 + 2x + x^2 + 5x^3) \times (-2 - x + 2x^2 - 2x^3)\]
\[(4 + 4x + 3x^2 + 9x^3) \times x^2\]

\[6 + 7x + 2x^2\]
\[-2 + 8x^2\]
\[5 + 21x + 18x^2\]

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

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

\[(3 + 2x + 2x^2 + 4x^3) \times (2 + x - x^2 + 2x^3)\]
\[(1 + 2x + x^2 + 5x^3) \times (-2 - x + 2x^2 - 2x^3)\]
\[(4 + 4x + 3x^2 + 9x^3) \times x^2\]

\[6 + 7x + 2x^2\]
\[-2 + 8x^2\]
\[5 + 21x + 18x^2\]

\[(3 + 2x) \times (2 + x)\]
\[(2 + 4x) \times (-1 + 2x)\]
\[(5 + 6x) \times (1 + 3x)\]

\[(5 + 21x + 18x^2) - (6 + 7x + 2x^2) - (-2 + 8x^2) = 1 + 14x + 8x^2\]
\[(6 + 7x + 2x^2) + (1 + 14x + 8x^2)x^2 + (-2 + 8x^2)x^4\]
\[= 6 + 7x + 3x^2 + 14x^3 + 6x^4 + 8x^6\]
Example

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

\[
(3 + 2x + 2x^2 + 4x^3)
\times(2 + x - x^2 + 2x^3)
\]

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

\[
(4 + 4x + 3x^2 + 9x^3)
\times x^2
\]

\[
(5 + 6x + 2x^2)
\times(1 + 3x)
\]

\[
(3 + 2x)
\times(2 + x)
\]

\[
(2 + 4x)
\times(-1 + 2x)
\]

\[
(5 + 6x)
\times(1 + 3x)
\]

\[
6 + 7x + 2x^2 + (1 + 14x + 8x^2)x^2 + (-2 + 8x^2)x^4
\]

\[
6 + 7x + 3x^2 + 14x^3 + 6x^4 + 8x^6
\]

\[
(5 + 21x + 18x^2) - (6 + 7x + 2x^2) - (-2 + 8x^2) = 1 + 14x + 8x^2
\]
Example

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

\[(3 + 2x + 2x^2 + 4x^3) \times (2 + x - x^2 + 2x^3)\]

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

\[(4 + 4x + 3x^2 + 9x^3) \times x^2\]

\[6 + 7x + 2x^2\]

\[-2 + 8x^2\]

\[5 + 21x + 18x^2\]

\[6 + 7x + 2x^2\]

\[-2 + 8x^2\]

\[5 + 21x + 18x^2\]

\[0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6\]
\[
\begin{array}{cccccccc}
6 & 7 & 2 & -2 & 0 & 8 & & \\
\end{array}
\]

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

\[(6 + 7x + 2x^2) + (1 + 14x + 8x^2)x^2 + (-2 + 8x^2)x^4\]
\[= 6 + 7x + 3x^2 + 14x^3 + 6x^4 + 8x^6\]
Example

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

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

\[
(6 + 7x + 2x^2) + (1 + 14x + 8x^2)x^2 + (-2 + 8x^2)x^4
\]

\[= 6 + 7x + 3x^2 + 14x^3 + 6x^4 + 8x^6\]
Example

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

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

\[(6 + 7x + 2x^2) - (2 + 8x^2) = 1 + 14x + 8x^2\]

\[(6 + 7x + 2x^2) + (1 + 14x + 8x^2)x^2 + (-2 + 8x^2)x^4 = 6 + 7x + 3x^2 + 14x^3 + 6x^4 + 8x^6\]
Example

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

\[
\begin{array}{c}
6 + 7x + 3x^2 + 14x^3 + 6x^4 + 8x^6 \\
(3 + 2x + 2x^2 + 4x^3) \times (2 + x - x^2 + 2x^3) \\
6 + 7x + 2x^2 \\
(3 + 2x) \times (2 + x) \\
(5 + 6x) \times (1 + 3x) \\
(1 + 2x + x^2 + 5x^3) \times (-2 - x + 2x^2 - 2x^3) \\
-2 + 8x^2 \\
(2 + 4x) \times (-1 + 2x) \\
(4 + 4x + 3x^2 + 9x^3) \times x^2 \\
5 + 21x + 18x^2 \\
(5 + 6x) + (1 + 3x) \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 7 & 2 & -2 & 0 & 8 \\
6 & 7 & 3 & 14 & 6 & 0 & 8 \\
\end{array}
\]

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

\[
(6 + 7x + 2x^2) + (1 + 14x + 8x^2)x^2 + (-2 + 8x^2)x^4
\]

\[
= 6 + 7x + 3x^2 + 14x^3 + 6x^4 + 8x^6
\]
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:** $C = AB$

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

1: for $i \leftarrow 1$ to $n$
2:     for $j \leftarrow 1$ to $n$
3:         $C[i, j] \leftarrow 0$
4:     for $k \leftarrow 1$ to $n$
5:         $C[i, j] \leftarrow C[i, j] + A[i, k] \times B[k, j]$
6: return $C$

running time = $O(n^3)$
Matrix Multiplication

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

**Output:** $C = AB$

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

1: for $i \leftarrow 1$ to $n$ do
2:     for $j \leftarrow 1$ to $n$ do
3:         $C[i, j] \leftarrow 0$
4:     for $k \leftarrow 1$ to $n$ do
5:         $C[i, j] \leftarrow C[i, j] + A[i, k] \times B[k, j]$
6: return $C$

**running time** = $O(n^3)$
Matrix Multiplication

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

**Output:** $C = AB$

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

1. **for** $i \leftarrow 1$ to $n$ **do**
   2. **for** $j \leftarrow 1$ to $n$ **do**
   3. $C[i, j] \leftarrow 0$
   4. **for** $k \leftarrow 1$ to $n$ **do**
   5. $C[i, j] \leftarrow C[i, j] + A[i, k] \times B[k, j]$

6. **return** $C$

- running time = $O(n^3)$
Try to Use Divide-and-Conquer

\[ A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \]

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

- matrix_multiplication\((A, B)\) recursively calls matrix_multiplication\((A_{11}, B_{11})\), matrix_multiplication\((A_{12}, B_{21})\), \ldots
Try to Use Divide-and-Conquer

\[ A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \]

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

matrix\_multiplication(A, B) recursively calls matrix\_multiplication(A_{11}, B_{11}), matrix\_multiplication(A_{12}, B_{21}), \ldots

Recurrence for running time: \( T(n) = 8T(n/2) + O(n^2) \)

\( T(n) = O(n^3) \)
Try to Use Divide-and-Conquer

\[
A = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

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

- matrix_multiplication\((A, B)\) recursively calls
  - matrix_multiplication\((A_{11}, B_{11})\),
  - matrix_multiplication\((A_{12}, B_{21})\),
  - \ldots

- Recurrence for running time:
  \[
  T(n) = 8T(n/2) + O(n^2)
  \]

- \(T(n) = O(n^3)\)

- Strassen’s Algorithm: 
  \[
  T(n) = 7T(n/2) + O(n^2)
  \]
Try to Use Divide-and-Conquer

\[
A = \begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}
\]
\[
B = \begin{pmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{pmatrix}
\]

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

matrix_multiplication(A, B) recursively calls
matrix_multiplication(A_{11}, B_{11}), matrix_multiplication(A_{12}, B_{21}), \ldots

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

\[ A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \]

\[ C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix} \]
Strassen’s Algorithm

\[
A = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \quad B = \begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

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

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

- \(C_{11} \leftarrow M_1 + M_4 - M_5 + M_7\)
- \(C_{12} \leftarrow M_3 + M_5\)
- \(C_{21} \leftarrow M_2 + M_4\)
- \(C_{22} \leftarrow M_1 - M_2 + M_3 + M_6\)
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. \textbf{FFT (Fast Fourier Transform): Polynomial Multiplication in} \(O(n \log n)\) \textbf{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

- \( p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} \)
- Known: given the value of \( p(x) \) for \( n \) different values of \( x \), \( p \) is uniquely determined

\[
\begin{pmatrix}
1 & 0 & 0 \\
1 & 1 & 1 \\
1 & 2 & 4 \\
\end{pmatrix}
- \begin{pmatrix}
1 \\
-1 \\
2 \\
\end{pmatrix}
= \begin{pmatrix}
1 & 0 & 0 \\
-3 & 2 & 2 \\
-1 & 2 & 7 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 \\
2 \\
7 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 0 & 0 \\
1 & 1 & 1 \\
1 & 2 & 4 \\
\end{pmatrix}
- \begin{pmatrix}
1 \\
-1 \\
2 \\
\end{pmatrix}
= \begin{pmatrix}
1 & 0 & 0 \\
3 & -2 & -2 \\
1 & -2 & -1 \\
\end{pmatrix}
\]

\[
\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
- \( p(x) = 1 - x + 2 x^2 : p(0) = 1, p(1) = 2, p(2) = 7. \)

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

- \( p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} \)

- 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

\[ p(x) = 1 - x + 2x^2, \quad q(x) = 3 - x^2 \]

Interpolation on 5 points \{0, 1, 2, 3, 4\}:

Interpolation for \( p \):

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
1 & 2 & 4 & 8 & 16 \\
1 & 3 & 9 & 27 & 81 \\
1 & 4 & 16 & 64 & 256 \\
\end{pmatrix}
\begin{pmatrix}
1 \\
-1 \\
2 \\
0 \\
0 \\
\end{pmatrix}
= \begin{pmatrix}
1 \\
2 \\
7 \\
16 \\
29 \\
\end{pmatrix}
\]

Interpolation for \( q \):

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
1 & 2 & 4 & 8 & 16 \\
1 & 3 & 9 & 27 & 81 \\
1 & 4 & 16 & 64 & 256 \\
\end{pmatrix}
\begin{pmatrix}
3 \\
0 \\
-1 \\
0 \\
0 \\
\end{pmatrix}
= \begin{pmatrix}
3 \\
2 \\
-1 \\
-6 \\
-13 \\
\end{pmatrix}
\]
Using Interpolation for Polynomial Multiplication

\[ p(x) = 1 - x + 2x^2, \quad q(x) = 3 - x^2 \]
Using Interpolation for Polynomial Multiplication

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

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
1 & 2 & 4 & 8 & 16 \\
1 & 3 & 9 & 27 & 81 \\
1 & 4 & 16 & 64 & 256
\end{pmatrix} \times \begin{pmatrix}
1 \\
-1 \\
2 \\
0 \\
0
\end{pmatrix} = \begin{pmatrix}
1 \\
7 \\
16 \\
29
\end{pmatrix}
\]

interpolation for \( p \):

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
1 & 2 & 4 & 8 & 16 \\
1 & 3 & 9 & 27 & 81 \\
1 & 4 & 16 & 64 & 256
\end{pmatrix} \times \begin{pmatrix}
3 \\
0 \\
-1 \\
0 \\
0
\end{pmatrix} = \begin{pmatrix}
3 \\
2 \\
-1 \\
-6 \\
-13
\end{pmatrix}
\]

interpolation for \( q \):
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 \( x_0, x_1, x_2, \ldots, x_{m-1} \) carefully, \( m = 2n - 1 \)
- Compute the interpolation of \( p \) and \( q \):

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

\[
M \begin{pmatrix}
a_0 \\
a_1 \\
\vdots \\
a_{n-1} \\
o
\end{pmatrix} =
M \begin{pmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{m-1}
\end{pmatrix}
\]

\[
M \begin{pmatrix}
b_0 \\
b_1 \\
\vdots \\
b_{n-1} \\
o
\end{pmatrix} =
M \begin{pmatrix}
z_0 \\
z_1 \\
z_2 \\
\vdots \\
z_{m-1}
\end{pmatrix}
\]
Multiplication of two polynomials of degree $n - 1$

\[
M \begin{pmatrix}
    c_0 \\
    c_1 \\
    \vdots \\
    c_{m-1}\end{pmatrix} = \begin{pmatrix}
    y_0z_0 \\
    y_1z_1 \\
    y_2z_2 \\
    \vdots \\
    y_{m-1}z_{m-1}\end{pmatrix} = M^{-1} \begin{pmatrix}
    y_0z_0 \\
    y_1z_1 \\
    y_2z_2 \\
    \vdots \\
    y_{m-1}z_{m-1}\end{pmatrix}
\]
Multiplication of two polynomials of degree $n - 1$

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

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

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

- $\omega := e^{\frac{2\pi i}{n}}$, $n$-th roots are $1, \omega, \omega^2, \cdots, \omega^{n-1}$
\[ F_n := \begin{pmatrix}
1 & 1 & 1 & 1 & \cdots & 1 \\
1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\
1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)}
\end{pmatrix} \]

- **Interpolation and Inverse-Interpolation:**

\[
\begin{pmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{pmatrix} = F_n \begin{pmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{pmatrix} = F_n^{-1} \begin{pmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{pmatrix}
\]
\[ F_n := \begin{pmatrix}
1 & 1 & 1 & 1 & \cdots & 1 \\
1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\
1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)}
\end{pmatrix} \]

- **Interpolation and Inverse-Interpolation:**

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

- **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

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

**Breaking polynomial into even and odd parts**

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

$$p(\omega^k) = p_{\text{even}}(\omega^{2k}) + p_{\text{odd}}(\omega^{2k}) \cdot \omega^k, \quad k = 0, 1, \cdots, \frac{n}{2} - 1$$

$$p(\omega^{n/2+k}) = p_{\text{even}}(\omega^{2k}) - p_{\text{odd}}(\omega^{2k}) \cdot \omega^k, \quad k = 0, 1, \cdots, \frac{n}{2} - 1$$
Assume $n$ is an integer power of 2

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

1: **if** $n = 1$ **then return** $(a_0)$

2: $(e_0, e_1, \cdots, e_{n/2-1}) \leftarrow \text{FFT}(n/2, a_0, a_2, \cdots, a_{n-2})$

3: $(o_0, o_1, \cdots, o_{n/2-1}) \leftarrow \text{FFT}(n/2, a_1, a_3, \cdots, a_{n-1})$

4: **for** $k \leftarrow 0, 1, 2, \cdots n/2 - 1$ **do**

5: $y_k \leftarrow e_k + o_k \cdot \omega^k$

6: $y_{n/2+k} \leftarrow e_k - o_k \cdot \omega^k$

7: **return** $(y_0, y_1, \cdots, y_{n-1})$
Assume $n$ is an integer power of 2

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

1. **if** $n = 1$ **then return** $(a_0)$
2. $(e_0, e_1, \cdots, e_{n/2-1}) \leftarrow \text{FFT}(n/2, a_0, a_2, \cdots, a_{n-2})$
3. $(o_0, o_1, \cdots, o_{n/2-1}) \leftarrow \text{FFT}(n/2, a_1, a_3, \cdots, a_{n-1})$
4. **for** $k \leftarrow 0, 1, 2, \cdots n/2 - 1$ **do**
   5. $y_k \leftarrow e_k + o_k \cdot \omega^k$
   6. $y_{n/2+k} \leftarrow e_k - o_k \cdot \omega^k$
7. **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

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

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

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

\[\omega = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}\]
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

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

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

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

\[\omega = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}\]

\[y_0 = e_0 + o_0 = 10 + 14 = 24\]

\[y_1 = e_1 + o_1\omega = -2 + (-4 - 2i)\left(\frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}\right) = -2 - 2\sqrt{2} - 3\sqrt{2}i\]
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

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

\[
\begin{pmatrix}
e_0 \\
e_1 \\
e_2 \\
e_3 \\
o_0 \\
o_1 \\
o_2 \\
o_3
\end{pmatrix} = \begin{pmatrix}1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix}3 \\ 1 \\ 5 \\ 1 \end{pmatrix} = \begin{pmatrix}10 \\ -2 \\ 6 \\ -2 \end{pmatrix}
\]

\[
\omega = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}
\]

\[
y_0 = e_0 + o_0 = 10 + 14 = 24
\]

\[
y_1 = e_1 + o_1 \omega = -2 + (-4 - 2i)\left(\frac{\sqrt{2}}{2} + \frac{\sqrt{2}i}{2}\right) = -2 - 2\sqrt{2} - 3\sqrt{2}i
\]

\[
y_6 = e_2 - o_2 \omega^2 = 6 - 2i
\]

\[
y_7 = e_3 - o_3 \omega^3
\]
\[ p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} \]

\[ q(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1} \]

Multiplying \( p \) and \( q \), assuming \( n \) is a power of 2:

1: \( y \leftarrow \text{FFT}(2n, a_0, a_1, \cdots, a_{n-1}, 0, 0, \cdots, 0) \)
2: \( z \leftarrow \text{FFT}(2n, b_0, b_1, \cdots, b_{n-1}, 0, 0, \cdots, 0) \)
3: \( c \leftarrow \text{iFFT}(2n, y_0 z_0, y_1 z_1, \cdots, y_{2n-1} z_{2n-1}) \)
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

\[
\frac{1}{n} \begin{pmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\
1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega & \omega^2 & \cdots & \omega^{n-1}
\end{pmatrix}
\]
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: $(x_1, y_1), (x_2, y_2), \ldots , (x_n, y_n)$

**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

![Diagram of points divided by a vertical line](image)
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

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
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
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: $T(n) = O(n \log^2 n)$

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
- 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

- $T(n) = 2T(n/2) + O(n) \implies T(n) = O(n \log n)$
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))
- CP(1, 8, (5, 7, 1, 3, 4, 8, 2, 6))
- CP(1, 4, (1, 3, 4, 2))
- CP(9, 16, (16, 9, 15, 14, 12, 13, 10, 11))
Example for Closest Pair

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

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

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))
Example for Closest Pair

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

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))
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))

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))
Example for Closest Pair

![Diagram of closest pair example]
Example for Closest Pair
Example for Closest Pair
Example for Closest Pair

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

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

CP(1, 8), (5, 7), (1, 3), (4, 8), (2, 6)

CP(1, 4), (1, 3), (4, 2)

CP(9, 16), (16, 9), (15, 14), (12, 13), (10, 11)
Example for Closest Pair

\[ \text{CP}(1, 16, (5, 16, 9, 15, 7, 14, 1, 12, 3, 4, 8, 13, 10, 11, 2, 6)) \]
Example for Closest Pair

- $\text{CP}(1, 16, (5, 16, 9, 15, 7, 14, 1, 12, 3, 4, 8, 13, 10, 11, 2, 6))$
- $\text{CP}(1, 8, (5, 7, 1, 3, 4, 8, 2, 6))$
Example for Closest Pair

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

Diagram showing points labeled from 1 to 16 with close pairs identified.
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, ...
Computing $F_n$ : Stupid Divide-and-Conquer Algorithm

Fib($n$)

1: if $n = 0$ return 0
2: if $n = 1$ return 1
3: 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($n$)

1: if $n = 0$ return 0
2: if $n = 1$ return 1
3: 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**($n$)

1: if $n = 0$ return 0
2: if $n = 1$ return 1
3: 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

\[
\text{Fib}(n) =
\begin{align*}
1: & \text{ if } n = 0 \text{ return } 0 \\
2: & \text{ if } n = 1 \text{ return } 1 \\
3: & \text{ return Fib}(n - 1) + \text{Fib}(n - 2)
\end{align*}
\]

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($n$)**

1. $F[0] \leftarrow 0$
2. $F[1] \leftarrow 1$
3. for $i \leftarrow 2$ to $n$ do
4. \hspace{1em} $F[i] \leftarrow F[i - 1] + F[i - 2]$
5. return $F[n]$

- Dynamic Programming
Computing $F_n$: Reasonable Algorithm

Fib($n$)

1. $F[0] \leftarrow 0$
2. $F[1] \leftarrow 1$
3. for $i \leftarrow 2$ to $n$ do
   4. $F[i] \leftarrow F[i - 1] + F[i - 2]$
5. return $F[n]$
Computing $F_n$: Reasonable Algorithm

Fib($n$)

1: $F[0] \leftarrow 0$
2: $F[1] \leftarrow 1$
3: for $i \leftarrow 2$ to $n$ do
   4: \hspace{1em} $F[i] \leftarrow F[i - 1] + F[i - 2]$
5: return $F[n]$

• Dynamic Programming
• Running time = $O(n)$
Computing $F_n$: Even Better Algorithm

\[
\begin{pmatrix}
F_n \\
F_{n-1}
\end{pmatrix}
= 
\begin{pmatrix}
1 & 1 \\
1 & 0
\end{pmatrix}
\begin{pmatrix}
F_{n-1} \\
F_{n-2}
\end{pmatrix}
\]

\[
\begin{pmatrix}
F_n \\
F_{n-1}
\end{pmatrix}
= 
\begin{pmatrix}
1 & 1 \\
1 & 0
\end{pmatrix}^2
\begin{pmatrix}
F_{n-2} \\
F_{n-3}
\end{pmatrix}
\]

\[
\begin{pmatrix}
F_n \\
F_{n-1}
\end{pmatrix}
= 
\begin{pmatrix}
1 & 1 \\
1 & 0
\end{pmatrix}^{n-1}
\begin{pmatrix}
F_1 \\
F_0
\end{pmatrix}
\]
**power($n$)**

1. if $n = 0$ then return \[
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]
2. $R \leftarrow \text{power}([n/2])$
3. $R \leftarrow R \times R$
4. if $n$ is odd then $R \leftarrow R \times \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
5. return $R$

**Fib($n$)**

1. if $n = 0$ then return 0
2. $M \leftarrow \text{power}(n - 1)$
3. return $M[1][1]$
**power(n)**

1: if $n = 0$ then return \[
\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}
\]
2: $R \leftarrow \text{power}(\lfloor n/2 \rfloor)$
3: $R \leftarrow R \times R$
4: if $n$ is odd then $R \leftarrow R \times \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
5: return $R$

**Fib(n)**

1: if $n = 0$ then return 0
2: $M \leftarrow \text{power}(n - 1)$
3: return $M[1][1]$

- Recurrence for running time?
power\( (n) \)

1: if \( n = 0 \) then return \( \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \)

2: \( R \leftarrow \text{power}(\lfloor n/2 \rfloor) \)

3: \( R \leftarrow R \times R \)

4: if \( n \) is odd then \( R \leftarrow R \times \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \)

5: return \( R \)

Fib\( (n) \)

1: if \( n = 0 \) then return 0

2: \( M \leftarrow \text{power}(n - 1) \)

3: return \( M[1][1] \)

● Recurrence for running time? \( T(n) = T(n/2) + O(1) \)
**power(n)**

1. if \( n = 0 \) then return \[
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]

2. \( R \leftarrow \text{power}\left(\lfloor n/2 \rfloor \right) \)

3. \( R \leftarrow R \times R \)

4. if \( n \) is odd then \( R \leftarrow R \times \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \)

5. return \( R \)

**Fib(n)**

1. if \( n = 0 \) then return 0

2. \( M \leftarrow \text{power}(n - 1) \)

3. 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!

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

A: $\Theta(n)$

We cannot 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.
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 cannot 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.
Running time $= O(\log n)$: We Cheated!

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

A: $\Theta(n)$

- We cannot 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, ...:
  \[ T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n) \]
Summary: Divide-and-Conquer

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

- 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, \ldots:
  \[ T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n) \]
- 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, ···:
  \[ T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n) \]

- 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, ...