算法设计与分析(2024年春季学期)
Introduction and Syllabus

授课老师：栗师
南京大学计算机科学与技术系
Outline

1 Syllabus

2 Introduction
   - What is an Algorithm?
   - Example: Insertion Sort
   - Analysis of Insertion Sort

3 Asymptotic Notations

4 Common Running times
Course Webpage:
https://tcs.nju.edu.cn/shili/courses/2024spring-algo
1 Syllabus

2 Introduction
- What is an Algorithm?
- Example: Insertion Sort
- Analysis of Insertion Sort

3 Asymptotic Notations

4 Common Running times
Outline

1. Syllabus

2. Introduction
   - What is an Algorithm?
   - Example: Insertion Sort
   - Analysis of Insertion Sort

3. Asymptotic Notations

4. Common Running times
What is an Algorithm?

- Donald Knuth: An algorithm is a finite, definite effective procedure, with some input and some output.
- Computational problem: specifies the input/output relationship.
- An algorithm solves a computational problem if it produces the correct output for any given input.
Examples

Greatest Common Divisor

**Input:** two integers \( a, b > 0 \)

**Output:** the greatest common divisor of \( a \) and \( b \)

Example:

- **Input:** 210, 270
- **Output:** 30

**Algorithm:** Euclidean algorithm

\[
gcd(270, 210) = gcd(210, 270 \mod 210) = gcd(210, 60)
\]

\[
(270, 210) \rightarrow (210, 60) \rightarrow (60, 30) \rightarrow (30, 0)
\]
Examples

### Sorting

**Input:** sequence of \( n \) numbers \((a_1, a_2, \ldots, a_n)\)

**Output:** a permutation \((a'_1, a'_2, \ldots, a'_n)\) of the input sequence such that \(a'_1 \leq a'_2 \leq \cdots \leq a'_n\)

Example:
- **Input:** 53, 12, 35, 21, 59, 15
- **Output:** 12, 15, 21, 35, 53, 59

*Algorithms:* insertion sort, merge sort, quicksort, \ldots
Examples

**Shortest Path**

**Input:** directed graph $G = (V, E), s, t \in V$

**Output:** a shortest path from $s$ to $t$ in $G$

Algorithm: Dijkstra’s algorithm
Algorithm = Computer Program?

- Algorithm: “abstract”, can be specified using computer program, English, pseudo-codes or flow charts.
- Computer program: “concrete”, implementation of algorithm, using a particular programming language
Pseudo-Code

Euclidean($a, b$)
1: while $b > 0$ do
2: \[(a, b) \leftarrow (b, a \mod b)\]
3: return $a$

C++ program:

```cpp
int Euclidean(int a, int b) {
    int c;
    while (b > 0) {
        c = b;
        b = a % b;
        a = c;
    }
    return a;
}
```
Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage
- Not covered in the course: engineering side
  - extensibility
  - modularity
  - object-oriented model
  - user-friendliness (e.g., GUI)
  - ...

Why is it important to study the running time (efficiency) of an algorithm?

1. feasible vs. infeasible
2. efficient algorithms: less engineering tricks needed, can use languages aiming for easy programming (e.g., python)
3. fundamental
4. it is fun!
Outline

1 Syllabus

2 Introduction
   - What is an Algorithm?
   - Example: Insertion Sort
   - Analysis of Insertion Sort

3 Asymptotic Notations

4 Common Running times
Sorting Problem

**Input:** sequence of \( n \) numbers \((a_1, a_2, \cdots, a_n)\)

**Output:** a permutation \((a'_1, a'_2, \cdots, a'_n)\) of the input sequence such that \(a'_1 \leq a'_2 \leq \cdots \leq a'_n\)

**Example:**

- **Input:** 53, 12, 35, 21, 59, 15
- **Output:** 12, 15, 21, 35, 53, 59
Insertion-Sort

- At the end of $j$-th iteration, the first $j$ numbers are sorted.

iteration 1: 53, 12, 35, 21, 59, 15
iteration 2: 12, 53, 35, 21, 59, 15
iteration 3: 12, 35, 53, 21, 59, 15
iteration 4: 12, 21, 35, 53, 59, 15
iteration 5: 12, 21, 35, 53, 59, 15
iteration 6: 12, 15, 21, 35, 53, 59
Example:

- **Input**: 53, 12, 35, 21, 59, 15
- **Output**: 12, 15, 21, 35, 53, 59

**insertion-sort**($A, n$)

1: **for** $j \leftarrow 2$ to $n$ **do**
2:     $key \leftarrow A[j]$
3:     $i \leftarrow j - 1$
4:     **while** $i > 0$ and $A[i] > key$ **do**
5:         $A[i + 1] \leftarrow A[i]$
6:     $i \leftarrow i - 1$
7:     $A[i + 1] \leftarrow key$

- $j = 6$
- $key = 15$

```
12 15 21 35 53 59
```

↑

```
i
```
Outline

1 Syllabus

2 Introduction
   - What is an Algorithm?
   - Example: Insertion Sort
   - Analysis of Insertion Sort

3 Asymptotic Notations

4 Common Running times
Analysis of Insertion Sort

- Correctness
- Running time
Invariant: after iteration $j$ of outer loop, $A[1..j]$ is the sorted array for the original $A[1..j]$.

- After $j = 1$: 53, 12, 35, 21, 59, 15
- After $j = 2$: 12, 53, 35, 21, 59, 15
- After $j = 3$: 12, 35, 53, 21, 59, 15
- After $j = 4$: 12, 21, 35, 53, 59, 15
- After $j = 5$: 12, 21, 35, 53, 59, 15
- After $j = 6$: 12, 15, 21, 35, 53, 59
Analyzing Running Time of Insertion Sort

Q1: what is the size of input?
A1: Running time as the function of size
possible definition of size:
- Sorting problem: \# integers,
- Greatest common divisor: total length of two integers
- Shortest path in a graph: \# edges in graph

Q2: Which input?
- For the insertion sort algorithm: if input array is already sorted in ascending order, then algorithm runs much faster than when it is sorted in descending order.

A2: Worst-case analysis:
- Running time for size $n = \text{worst running time over all possible arrays of length } n$
Analyzing Running Time of Insertion Sort

- Q3: How fast is the computer?
- Q4: Programming language?
- A: They do not matter!

**Important idea: asymptotic analysis**
- Focus on growth of running-time as a function, not any particular value.
Asymptotic Analysis: \( O \)-notation

Informal way to define \( O \)-notation:
- Ignoring lower order terms
- Ignoring leading constant

\[
3n^3 + 2n^2 - 18n + 1028 \Rightarrow 3n^3 \Rightarrow n^3 \\
3n^3 + 2n^2 - 18n + 1028 = O(n^3) \\
\]
\[
n^2/100 - 3n + 10 \Rightarrow n^2/100 \Rightarrow n^2 \\
n^2/100 - 3n + 10 = O(n^2) \\
\]
Asymptotic Analysis: $O$-notation

- $3n^3 + 2n^2 - 18n + 1028 = O(n^3)$
- $n^2/100 - 3n^2 + 10 = O(n^2)$

$O$-notation allows us to ignore
- architecture of computer
- programming language
- how we measure the running time: seconds or # instructions?
- to execute $a \leftarrow b + c$:
  - program 1 requires 10 instructions, or $10^{-8}$ seconds
  - program 2 requires 2 instructions, or $10^{-9}$ seconds
  - they only change by a constant in the running time, which will be hidden by the $O(\cdot)$ notation
Algorithm 1 runs in time $O(n^2)$
Algorithm 2 runs in time $O(n)$

Does not tell which algorithm is faster for a specific $n$!
Algorithm 2 will eventually beat algorithm 1 as $n$ increases.

For Algorithm 1: if we increase $n$ by a factor of 2, running time increases by a factor of 4
For Algorithm 2: if we increase $n$ by a factor of 2, running time increases by a factor of 2
Asymptotic Analysis of Insertion Sort

insertion-sort\((A, n)\)

1: \textbf{for } \ j \leftarrow 2 \textbf{ to } n \textbf{ do}
2: \hspace{1em} key \leftarrow A[j]
3: \hspace{1em} i \leftarrow j - 1
4: \hspace{1em} \textbf{while } i > 0 \text{ and } A[i] > key \textbf{ do}
5: \hspace{2em} A[i + 1] \leftarrow A[i]
6: \hspace{1em} i \leftarrow i - 1
7: \hspace{1em} A[i + 1] \leftarrow key

- Worst-case running time for iteration \(j\) of the outer loop?
  
  Answer: \(O(j)\)

- Total running time = \(\sum_{j=2}^{n} O(j) = O(\sum_{j=2}^{n} j)\)
  
  \[= O\left(\frac{n(n+1)}{2} - 1\right) = O(n^2)\]
Computation Model

- Random-Access Machine (RAM) model
  - reading and writing \( A[j] \) takes \( O(1) \) time
- Basic operations such as addition, subtraction and multiplication take \( O(1) \) time
- Each integer (word) has \( c \log n \) bits, \( c \geq 1 \) large enough
  - Reason: often we need to read the integer \( n \) and handle integers within range \([-n^c, n^c]\), it is convenient to assume this takes \( O(1) \) time.
- What is the precision of real numbers?
  Most of the time, we only consider integers.
- Can we do better than insertion sort asymptotically?
  Yes: merge sort, quicksort and heap sort take \( O(n \log n) \) time
Questions?
Outline

1. Syllabus

2. Introduction
   - What is an Algorithm?
   - Example: Insertion Sort
   - Analysis of Insertion Sort

3. Asymptotic Notations

4. Common Running times
Asymptotically Positive Functions

**Def.** \( f : \mathbb{N} \rightarrow \mathbb{R} \) is an **asymptotically positive function** if:

- \( \exists n_0 > 0 \) such that \( \forall n > n_0 \) we have \( f(n) > 0 \)

- In other words, \( f(n) \) is positive for large enough \( n \).
- \( n^2 - n - 30 \) \hspace{1cm} Yes
- \( 2^n - n^{20} \) \hspace{1cm} Yes
- \( 100n - n^2 / 10 + 50 \) \hspace{1cm} No

- We only consider asymptotically positive functions.
**$O$-Notation: Asymptotic Upper Bound**

**$O$-Notation** For a function $g(n)$,

$$O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}.$$

- In short, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c > 0$ and every large enough $n$. 

![Graph showing $f(n) = O(g(n))$ and $cg(n)$]

\[ n_0 \]

\[ n \]
**O-Notation** For a function $g(n)$,

$$O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}.$$ 

- $3n^2 + 2n \in O(n^2 - 10n)$

**Proof.**

Let $c = 4$ and $n_0 = 50$, for every $n > n_0 = 50$, we have,

$$3n^2 + 2n - c(n^2 - 10n) = 3n^2 + 2n - 4(n^2 - 10n)$$

$$= -n^2 + 42n \leq 0.$$ 

$$3n^2 + 2n \leq c(n^2 - 10n)$$
**$O$-Notation**  For a function $g(n)$,

$$O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}.$$  

- $3n^2 + 2n \in O(n^2 - 10n)$
- $3n^2 + 2n \in O(n^3 - 5n^2)$
- $n^{100} \in O(2^n)$
- $n^3 \notin O(10n^2)$

<table>
<thead>
<tr>
<th>Asymptotic Notations</th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Relations</td>
<td>$\leq$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conventions

- We use “\( f(n) = O(g(n)) \)” to denote “\( f(n) \in O(g(n)) \)”
- \( 3n^2 + 2n = O(n^2) \)
- “\( = \)” is asymmetric: we do not write \( O(n^2) = 3n^2 + 2n \)
- Analogy: Mike is a student. A student is Mike.
- We use “\( O(g(n)) = O(g'(n)) \)” to denote “\( O(g(n)) \subseteq O(g'(n)) \)”.
- \( O(3n^2 + 2n) = O(n^2) \)
- Again, “\( = \)” is asymmetric.
- \( O(n^3) = O(3n^2 + 2n) \) makes sense, but is wrong.
- Analogy: All students are people.
- Equalities can be chained: \( 3n^2 + 2n = O(n^2) = O(n^3) \).
**Ω-Notation**: Asymptotic Lower Bound

**O-Notation**  For a function $g(n)$,

$$O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}.$$  

**Ω-Notation**  For a function $g(n)$,

$$\Omega(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \geq cg(n), \forall n \geq n_0 \}.$$  

- In short, $f(n) \in \Omega(g(n))$ if $f(n) \geq cg(n)$ for some $c$ and large enough $n$.  

**Ω-Notation** For a function $g(n)$,

$$\Omega(g(n)) = \left\{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \geq cg(n), \forall n \geq n_0 \right\}.$$
Ω-Notation: Asymptotic Lower Bound

- Again, we use “=" instead of ∈.
- \(4n^2 = \Omega(n - 10)\)
- \(3n^2 - n + 10 = \Omega(n^2 - 20)\)

<table>
<thead>
<tr>
<th>Asymptotic Notations</th>
<th>O</th>
<th>Ω</th>
<th>Θ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Relations</td>
<td>≤</td>
<td>≥</td>
<td></td>
</tr>
</tbody>
</table>

Theorem \(f(n) = O(g(n)) \iff g(n) = \Omega(f(n))\).
**Θ-Notation: Asymptotic Tight Bound**

**Θ-Notation**  For a function $g(n)$,

$$\Theta(g(n)) = \{ \text{function } f : \exists c_2 \geq c_1 > 0, n_0 > 0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}.$$  

- $f(n) = \Theta(g(n))$, then for large enough $n$, we have "$f(n) \approx g(n)$".
**Θ-Notation: Asymptotic Tight Bound**

**Θ-Notation**  For a function \( g(n) \),

\[
Θ(g(n)) = \{ \text{function } f : \exists c_2 \geq c_1 > 0, n_0 > 0 \text{ such that } \\
c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}. 
\]

- \( 3n^2 + 2n = Θ(n^2 - 20n) \)
- \( 2^{n/3+100} = Θ(2^{n/3}) \)

<table>
<thead>
<tr>
<th>Asymptotic Notations</th>
<th>( O )</th>
<th>( Ω )</th>
<th>( Θ )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Comparison Relations</strong></td>
<td>( ≤ )</td>
<td>( ≥ )</td>
<td>( = )</td>
</tr>
</tbody>
</table>

**Theorem**  \( f(n) = Θ(g(n)) \) if and only if

\( f(n) = O(g(n)) \) and \( f(n) = Ω(g(n)) \).
**o and ω-Notations**

**o-Notation** For a function $g(n)$,

$$o(g(n)) = \{\text{function } f : \forall c > 0, \exists n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0\}.$$

**ω-Notation** For a function $g(n)$,

$$ω(g(n)) = \{\text{function } f : \forall c > 0, \exists n_0 > 0 \text{ such that } f(n) \geq cg(n), \forall n \geq n_0\}.$$

Example:

- $3n^2 + 5n + 10 = o(n^2 \log n)$.
- $3n^2 + 5n + 10 = ω(n^2 / \log n)$.

<table>
<thead>
<tr>
<th>Asymptotic Notations</th>
<th>$O$</th>
<th>$Ω$</th>
<th>$Θ$</th>
<th>$o$</th>
<th>$ω$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Relations</td>
<td>$\leq$</td>
<td>$≥$</td>
<td>$=$</td>
<td>$&lt;$</td>
<td>$&gt;$</td>
</tr>
<tr>
<td>Asymptotic Notations</td>
<td>$O$</td>
<td>$\Omega$</td>
<td>$\Theta$</td>
<td>$o$</td>
<td>$\omega$</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----</td>
<td>---------</td>
<td>--------</td>
<td>-----</td>
<td>--------</td>
</tr>
<tr>
<td>Comparison Relations</td>
<td>$\leq$</td>
<td>$\geq$</td>
<td>$=$</td>
<td>$&lt;$</td>
<td>$&gt;$</td>
</tr>
</tbody>
</table>

**Facts on Comparison Relations**

- $a \leq b \iff b \geq a$
- $a = b \iff a \leq b \text{ and } a \geq b$
- $a < b \implies a \leq b$
- $a < b \iff b > a$

**Correct Analogies**

- $f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$
- $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n))$
- $f(n) = o(g(n)) \implies f(n) = O(g(n))$
- $f(n) = o(g(n)) \iff g(n) = \omega(f(n))$
Asymptotic Notations

<table>
<thead>
<tr>
<th></th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
<th>$o$</th>
<th>$\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Relations</td>
<td>$\leq$</td>
<td>$\geq$</td>
<td>$=$</td>
<td>$&lt;$</td>
<td>$&gt;$</td>
</tr>
</tbody>
</table>

Facts on Comparison Relations

- $a \leq b$ or $a \geq b$
- $a \leq b \iff a = b$ or $a < b$

Incorrect Analogies

- $f(n) = O(g(n))$ or $f(n) = \Omega(g(n))$
- $f(n) = O(g(n)) \iff f(n) = \Theta(g(n))$ or $f(n) = o(g(n))$
Incorrect Analogy

\[ f(n) = O(g(n)) \text{ or } f(n) = \Omega(g(n)) \]

\[ f(n) = n^2 \]

\[ g(n) = \begin{cases} 
1 & \text{if } n \text{ is odd} \\
 n^3 & \text{if } n \text{ is even}
\end{cases} \]
Recall: Informal way to define $O$-notation

- ignoring lower order terms: $3n^2 - 10n - 5 \rightarrow 3n^2$
- ignoring leading constant: $3n^2 \rightarrow n^2$
- $3n^2 - 10n - 5 = O(n^2)$

In the formal definition of $O(\cdot)$, nothing tells us to ignore lower order terms and leading constant.

- $3n^2 - 10n - 5 = O(5n^2 - 6n + 5)$ is correct, though weird
- $3n^2 - 10n - 5 = O(n^2)$ is the most natural since $n^2$ is the simplest term we can have inside $O(\cdot)$. 
Notice that $O$ denotes asymptotic upper bound

- $n^2 + 2n = O(n^3)$ is correct.
- The following sentence is correct: the running time of insertion sort is $O(n^4)$.
- Usually we say: The running time of insertion sort is $O(n^2)$ and the bound is tight.
- Also correct: the worst-case running time of insertion sort is $\Theta(n^2)$. 
2 Introduction
   • What is an Algorithm?
   • Example: Insertion Sort
   • Analysis of Insertion Sort

3 Asymptotic Notations

4 Common Running times
Computing the sum of \( n \) numbers

\[
\begin{align*}
\text{sum}(A, n) & \quad 1: \quad S \leftarrow 0 \\
                   & \quad 2: \quad \text{for } i \leftarrow 1 \text{ to } n \\
                   & \quad 3: \quad S \leftarrow S + A[i] \\
                   & \quad 4: \quad \text{return } S
\end{align*}
\]
$O(n)$ (Linear) Running Time

- Merge two sorted arrays

```
3 8 12 20 32 48
5 7 9 25 29
3 5 7 8 9 12 20 25 29 32 48
```
**O(n) (Linear) Running Time**

merge($B, C, n_1, n_2$)  

\[ B \text{ and } C \text{ are sorted, with length } n_1 \text{ and } n_2 \]

1. $A \leftarrow []; i \leftarrow 1; j \leftarrow 1$
2. while $i \leq n_1$ and $j \leq n_2$ do
3. \hspace{1em} if $B[i] \leq C[j]$ then
4. \hspace{2em} append $B[i]$ to $A$; $i \leftarrow i + 1$
5. \hspace{1em} else
6. \hspace{2em} append $C[j]$ to $A$; $j \leftarrow j + 1$
7. if $i \leq n_1$ then append $B[i..n_1]$ to $A$
8. if $j \leq n_2$ then append $C[j..n_2]$ to $A$
9. return $A$

Running time = $O(n)$ where $n = n_1 + n_2$. 
merge-sort$(A, n)$

1. if $n = 1$ then
2. return $A$
3. $B \leftarrow$ merge-sort$(A[1..\lceil n/2 \rceil], \lfloor n/2 \rfloor)$
4. $C \leftarrow$ merge-sort$(A[\lfloor n/2 \rfloor + 1..n], n - \lfloor n/2 \rfloor)$
5. return merge$(B, C, \lceil n/2 \rceil, n - \lfloor n/2 \rfloor)$
$O(n \log n)$ Running Time

- Merge-Sort

Each level takes running time $O(n)$

There are $O(\log n)$ levels

Running time $= O(n \log n)$
$O(n^2)$ (Quadratic) Running Time

**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

closet-pair($x, y, n$)

1: $bestd \leftarrow \infty$
2: for $i \leftarrow 1$ to $n - 1$ do
3: for $j \leftarrow i + 1$ to $n$ do
4: $d \leftarrow \sqrt{(x[i] - x[j])^2 + (y[i] - y[j])^2}$
5: if $d < bestd$ then
6: $besti \leftarrow i, bestj \leftarrow j, bestd \leftarrow d$
7: return $(besti, bestj)$

Closest pair can be solved in $O(n \log n)$ time!
Multiply two matrices of size $n \times n$

```
matrix-multiplication(A, B, n)
1: C ← matrix of size $n \times n$, with all entries being 0
2: for $i \leftarrow 1$ to $n$ do
3:     for $j \leftarrow 1$ to $n$ do
4:         for $k \leftarrow 1$ to $n$ do
5:             $C[i, k] \leftarrow C[i, k] + A[i, j] \times B[j, k]$
6: return $C$
```
Def. An independent set of a graph $G = (V, E)$ is a subset $S \subseteq V$ of vertices such that for every $u, v \in S$, we have $(u, v) \not\in E$. 

![Graph Diagram]
Beyond Polynomial Time: $2^n$

Maximum Independent Set Problem

**Input:** graph $G = (V, E)$

**Output:** the maximum independent set of $G$

```
max-independent-set(G = (V, E))
1: $R \leftarrow \emptyset$
2: for every set $S \subseteq V$ do
3:     $b \leftarrow$ true
4:     for every $u, v \in S$ do
5:         if $(u, v) \in E$ then $b \leftarrow$ false
6:     if $b$ and $|S| > |R|$ then $R \leftarrow S$
7: return $R$
```

Running time = $O(2^n n^2)$. 
Beyond Polynomial Time: $n!$

Hamiltonian Cycle Problem

**Input:** a graph with $n$ vertices

**Output:** a cycle that visits each node exactly once, or say no such cycle exists
Beyond Polynomial Time: $n!$

Hamiltonian($G = (V, E)$)

1. for every permutation $(p_1, p_2, \cdots, p_n)$ of $V$ do
2. \hspace{1em} $b \leftarrow$ true
3. \hspace{1em} for $i \leftarrow 1$ to $n - 1$ do
4. \hspace{2em} if $(p_i, p_{i+1}) \notin E$ then $b \leftarrow$ false
5. \hspace{1em} if $(p_n, p_1) \notin E$ then $b \leftarrow$ false
6. \hspace{1em} if $b$ then return $(p_1, p_2, \cdots, p_n)$
7. return “No Hamiltonian Cycle”

Running time = $O(n! \times n)$
$O(\log n)$ (Logarithmic) Running Time

- Binary search
  - Input: sorted array $A$ of size $n$, an integer $t$;
  - Output: whether $t$ appears in $A$.
- E.g, search 35 in the following array:

```
  3  8  10  25  29  37  38  42  46  52  59  61  63  75  79
```
**O(log n) (Logarithmic) Running Time**

Binary search
- Input: sorted array $A$ of size $n$, an integer $t$;
- Output: whether $t$ appears in $A$.

\[
\text{binary-search}(A, n, t) \\
1: \ i \leftarrow 1, \ j \leftarrow n \\
2: \ \textbf{while} \ i \leq j \ \textbf{do} \\
3: \quad k \leftarrow \lfloor (i + j)/2 \rfloor \\
4: \quad \textbf{if} \ A[k] = t \ \textbf{return} \ true \\
5: \quad \textbf{if} \ t < A[k] \ \textbf{then} \ j \leftarrow k - 1 \ \textbf{else} \ i \leftarrow k + 1 \\
6: \ \textbf{return} \ false \\
\]

Running time $= O(\log n)$
Comparing the Orders of Running Times

- Sort the functions from smallest to largest asymptotically:
  \[
  \log n, \ n, \ n^2, \ n \log n, \ n!, \ 2^n, \ e^n, \ n^n, \ \log(n!)
  \]

- \[
  \log n, \ n, \ \{n \log n, \ \log(n!)} \ n^2, \ 2^n, \ e^n, \ n!, \ n^n
  \]

- \[
  \log n = o(n), \ n = o(n \log n), \ n \log n = \Theta(\log(n!))
  \]

- \[
  \log(n!) = o(n^2), \ n^2 = o(2^n), \ 2^n = o(e^n)
  \]

- \[
  e^n = o(n!), \ n! = o(n^n)
  \]
Terminologies

When we talk about upper bounds:

- Logarithmic time: $O(\lg n)$
- Linear time: $O(n)$
- Quadratic time: $O(n^2)$
- Cubic time: $O(n^3)$
- Polynomial time: $O(n^k)$ for some constant $k$
- Exponential time: $O(c^n)$ for some $c > 1$
- Sub-linear time: $o(n)$
- Sub-quadratic time: $o(n^2)$

When we talk about lower bounds:

- Super-linear time: $\omega(n)$
- Super-quadratic time: $\omega(n^2)$
- Super-polynomial time: $\bigcap_{k>0} \omega(n^k) = n^{\omega(1)}$
Goal of Algorithm Design

- Design algorithms to minimize the order of the running time.

- Using asymptotic analysis allows us to ignore the leading constants and lower order terms.

- Makes our life much easier! (E.g., the leading constant depends on the implementation, compiler and computer architecture of computer.)
Q: Can constants really be ignored?

- e.g., how can we compare an algorithm with running time $0.1n^2$ with an algorithm with running time $1000n$?

A:

- Sometimes no
- For most natural and simple algorithms, constants are not so big.
- Algorithm with lower order running time beats algorithm with higher order running time for reasonably large $n$. 