CSE 431/531: Algorithm Analysis and Design (Spring 2019)

Introduction and Syllabus

Lecturer: Shi Li

Department of Computer Science and Engineering
University at Buffalo
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 (contains schedule, policies, homeworks and slides):
http://www.cse.buffalo.edu/~shil/courses/CSE531/

Please sign up course on Piazza via link on course webpage
- announcements, polls, asking/answering questions
- Time and location:
  - MoWeFr, 9:00-9:50am
  - Alumni 97

- Instructor:
  - Shi Li, shil@buffalo.edu
  - Office hours: TBD via poll

- TA
  - Alexander Stachnik, ajstachn@buffalo.edu
  - Office hours: TBD via poll
You should already know:
You should already know:

- **Mathematical Tools**
  - Mathematical inductions
  - Probabilities and random variables
You should already know:

- **Mathematical Tools**
  - Mathematical inductions
  - Probabilities and random variables

- **Basic data Structures**
  - Stacks, queues, linked lists
You should already know:

- **Mathematical Tools**
  - Mathematical inductions
  - Probabilities and random variables

- **Basic data Structures**
  - Stacks, queues, linked lists

- **Some Programming Experience**
  - C, C++, Java or Python
You Will Learn

- Classic algorithms for classic problems
  - Sorting
  - Shortest paths
  - Minimum spanning tree
You Will Learn

- Classic algorithms for classic problems
  - Sorting
  - Shortest paths
  - Minimum spanning tree

- How to analyze algorithms
  - Correctness
  - Running time (efficiency)
  - Space requirement
You Will Learn

- Classic algorithms for classic problems
  - Sorting
  - Shortest paths
  - Minimum spanning tree

- How to analyze algorithms
  - Correctness
  - Running time (efficiency)
  - Space requirement

- Meta techniques to design algorithms
  - Greedy algorithms
  - Divide and conquer
  - Dynamic programming
  - Linear Programming
You Will Learn

- Classic algorithms for classic problems
  - Sorting
  - Shortest paths
  - Minimum spanning tree

- How to analyze algorithms
  - Correctness
  - Running time (efficiency)
  - Space requirement

- Meta techniques to design algorithms
  - Greedy algorithms
  - Divide and conquer
  - Dynamic programming
  - Linear Programming

- NP-completeness
<table>
<thead>
<tr>
<th>Topic</th>
<th>Lectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3 lectures</td>
</tr>
<tr>
<td>Basic Graph Algorithms</td>
<td>3 lectures</td>
</tr>
<tr>
<td>Greedy Algorithms</td>
<td>6 lectures, 1 recitation</td>
</tr>
<tr>
<td>Divide and Conquer</td>
<td>6 lectures, 1 recitation</td>
</tr>
<tr>
<td>Dynamic Programming</td>
<td>6 lectures, 1 recitation</td>
</tr>
<tr>
<td></td>
<td>1 recitation for homeworks</td>
</tr>
<tr>
<td>Mid-Term Exam</td>
<td>Apr 7, 2019, Mon</td>
</tr>
<tr>
<td>NP-Completeness</td>
<td>6 lectures, 1 recitation</td>
</tr>
<tr>
<td>Linear Programming</td>
<td>4 lectures</td>
</tr>
<tr>
<td></td>
<td>2 recitations for homeworks</td>
</tr>
<tr>
<td>Final review, Q&amp;A</td>
<td></td>
</tr>
<tr>
<td>Final Exam</td>
<td>May 15 2019, Wed, 8:00am-11:00am</td>
</tr>
</tbody>
</table>
Textbook

Textbook (Highly Recommended):
- Algorithm Design, 1st Edition, by Jon Kleinberg and Eva Tardos

Other Reference Books
Highly recommended: read the correspondent sections from the textbook (or reference book) before classes

Slides and example problems for recitations will be posted online before class
Grading

- 40% for homeworks
  - 5 homeworks, 3 of which have programming tasks
- 60% for mid-term + final exams, score for two exams is
  \[
  \max\{M \times 20\% + F \times 40\%, M \times 30\% + F \times 30\%\}
  \]
  \[
  M, F \in [0, 100]
  \]
For Homeworks, You Are Allowed to

- Use course materials (textbook, reference books, lecture notes, etc)
- Post questions on Piazza
- Ask me or TAs for hints
- Collaborate with classmates
  - Think about each problem for enough time before discussions
  - **Must write down solutions on your own, in your own words**
  - Write down names of students you collaborated with
For Homeworks, You Are Not Allowed to

- Use external resources
  - Can’t Google or ask questions online for solutions
  - Can’t read posted solutions from other algorithm course webpages
- Copy solutions from other students
For Programming Problems

- Need to implement the algorithms by yourself
- Can not copy codes from others or the Internet
- We use Moss
  (https://theory.stanford.edu/~aiken/moss/) to detect similarity of programs
Late Policy

- You have 1 “late credit”, using it allows you to turn in a homework late for three days.
- With no special reasons, no other late submissions will be accepted.
- Mid-Term and Final Exam will be closed-book
- Per Departmental Policy on Academia Integrity Violations, penalty for AI violation is:
  - “F” for the course
  - may lose financial support
  - case will be recorded in department and university databases
Mid-Term and Final Exam will be closed-book

Per Departmental Policy on Academia Integrity Violations, penalty for AI violation is:
- “F” for the course
- may lose financial support
- case will be recorded in department and university databases

Questions?
Outline

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
Donald Knuth: An algorithm is a finite, definite effective procedure, with some input and some output.
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 $b$
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
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
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
- $\text{gcd}(270, 210) = \text{gcd}(210, 270 \mod 210) = \text{gcd}(210, 60)$
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 \((a_1, a_2, \cdots, a_n)\)

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

**Sorting**

**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
Examples

**Sorting**

**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

- Algorithms: insertion sort, merge sort, quicksort, . . .
### Shortest Path

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

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

**Shortest Path**

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

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

![Graph Diagram]

Algorithm: Dijkstra's algorithm
Examples

Shortest Path

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

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

Shortest Path

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

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

![Diagram of a directed graph]

- 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, associated with a particular programming language
Pseudo-Code

Euclidean\((a, b)\)
1. while \(b > 0\)
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;
}
```
Main focus: correctness, running time (efficiency)
Theoretical Analysis of Algorithms

- Main focus: correctness, running time (efficiency)
- Sometimes: memory usage
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!
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?
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
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)
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
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
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 $n$
2. \hspace{1em} key $\leftarrow A[j]$
3. \hspace{1em} $i \leftarrow j - 1$
4. \hspace{2em} while $i > 0$ and $A[i] > key$
5. \hspace{3em} $A[i + 1] \leftarrow A[i]$
6. \hspace{2em} $i \leftarrow i - 1$
7. \hspace{1em} $A[i + 1] \leftarrow key$
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 $n$
2. $key \leftarrow A[j]$
3. $i \leftarrow j - 1$
4. while $i > 0$ and $A[i] > key$
   1. $A[i + 1] \leftarrow A[i]$
   2. $i \leftarrow i - 1$
7. $A[i + 1] \leftarrow key$

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

12 21 35 53 59 15

$\uparrow$

$i$
**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 \(n\)
2. \(key \leftarrow A[j]\)
3. \(i \leftarrow j - 1\)
4. while \(i > 0\) and \(A[i] > key\)
5. \(A[i + 1] \leftarrow A[i]\)
6. \(i \leftarrow i - 1\)
7. \(A[i + 1] \leftarrow key\)

- \(j = 6\)
- \(key = 15\)

\[12 \quad 21 \quad 35 \quad 53 \quad 59 \quad 59\]
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 $n$
2.   $key \leftarrow A[j]$
3.   $i \leftarrow j - 1$
4.   while $i > 0$ and $A[i] > key$
5.     $A[i + 1] \leftarrow A[i]$
6.     $i \leftarrow i - 1$
7.     $A[i + 1] \leftarrow key$

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

$12 \ 21 \ 35 \ 53 \ 59 \ 59$

$↑$

↑

$i$
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 \(n\)
2. \(\text{key} \leftarrow A[j]\)
3. \(i \leftarrow j - 1\)
4. while \(i > 0 \) and \(A[i] > \text{key}\)
5. \(A[i + 1] \leftarrow A[i]\)
6. \(i \leftarrow i - 1\)
7. \(A[i + 1] \leftarrow \text{key}\)
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 $n$
2. \hspace{1em} $key \leftarrow A[j]$
3. \hspace{1em} $i \leftarrow j - 1$
4. while $i > 0$ and $A[i] > key$
5. \hspace{1em} $A[i + 1] \leftarrow A[i]$
6. \hspace{1em} $i \leftarrow i - 1$
7. \hspace{1em} $A[i + 1] \leftarrow key$

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

12 21 35 53 53 59

↑

$i$
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 $n$
2. $key \leftarrow A[j]$
3. $i \leftarrow j - 1$
4. while $i > 0$ and $A[i] > key$
5. $A[i + 1] \leftarrow A[i]$
6. $i \leftarrow i - 1$
7. $A[i + 1] \leftarrow key$

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

12 21 35 35 53 59

↑

↑

$i$
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 $n$
2. $key \leftarrow A[j]$
3. $i \leftarrow j - 1$
4. while $i > 0$ and $A[i] > key$
   - $A[i + 1] \leftarrow A[i]$
   - $i \leftarrow i - 1$
5. $A[i + 1] \leftarrow key$

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

12 21 35 35 53 59

↑

$i$
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 \(n\)
2. \(key \leftarrow A[j]\)
3. \(i \leftarrow j - 1\)
4. while \(i > 0\) and \(A[i] > key\)
5. \(A[i + 1] \leftarrow A[i]\)
6. \(i \leftarrow i - 1\)
7. \(A[i + 1] \leftarrow key\)

- \(j = 6\)
- \(key = 15\)

12 21 21 35 53 59

↑
i
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 $n$
2. \hspace{1em} key $\leftarrow A[j]$
3. \hspace{1em} $i \leftarrow j - 1$
4. while $i > 0$ and $A[i] > key$
5. \hspace{2em} $A[i + 1] \leftarrow A[i]$
6. \hspace{2em} $i \leftarrow i - 1$
7. \hspace{1em} $A[i + 1] \leftarrow key$

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

12 21 21 35 53 59

↑

$i$
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 \( n \)
2. \( key \leftarrow A[j] \)
3. \( i \leftarrow j - 1 \)
4. while \( i > 0 \) and \( A[i] > key \)
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
Q: Size of input?
Q: Size of input?
A: Running time as function of size
Q: Size of input?
A: Running time as function of size
possible definition of size: # integers, total length of integers, # vertices in graph, # edges in graph
Analyze Running Time of Insertion Sort

- Q: Size of input?
- A: Running time as function of size
  possible definition of size: \# integers, total length of integers, \# vertices in graph, \# edges in graph
- Q: Which input?

Worst-case analysis:
Worst running time over all input instances of a given size

Important idea: asymptotic analysis
Focus on growth of running-time as a function, not any particular value.
Q: Size of input?
A: Running time as function of size
possible definition of size: # integers, total length of integers, # vertices in graph, # edges in graph

Q: Which input?
A: Worst-case analysis:
  Worst running time over all input instances of a given size
Analyze Running Time of Insertion Sort

- Q: Size of input?
  - A: Running time as function of size
  - possible definition of size: # integers, total length of integers, # vertices in graph, # edges in graph

- Q: Which input?
  - A: Worst-case analysis:
    - Worst running time over all input instances of a given size

- Q: How fast is the computer?
Analyze Running Time of Insertion Sort

- Q: Size of input?
  - A: Running time as function of size

  possible definition of size: # integers, total length of integers, # vertices in graph, # edges in graph

- Q: Which input?
  - A: Worst-case analysis:
    - Worst running time over all input instances of a given size

- Q: How fast is the computer?

- Q: Programming language?
Q: Size of input?
A: Running time as function of size
possible definition of size: \# integers, total length of integers, \# vertices in graph, \# edges in graph

Q: Which input?
A: Worst-case analysis:
   Worst running time over all input instances of a given size

Q: How fast is the computer?
Q: Programming language?
A: Important idea: asymptotic analysis
   Focus on growth of running-time as a function, not any particular value.
Asymptotic Analysis: $O$-notation

- Ignoring lower order terms
- Ignoring leading constant
Asymptotic Analysis: $O$-notation

- Ignoring lower order terms
- Ignoring leading constant

$3n^3 + 2n^2 - 18n + 1028 \Rightarrow 3n^3 \Rightarrow n^3$
Asymptotic Analysis: \( O \)-notation

- Ignoring lower order terms
- Ignoring leading constant

\[
3n^3 + 2n^2 - 18n + 1028 \Rightarrow 3n^3 \Rightarrow n^3
\]

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

- Ignoring lower order terms
- Ignoring leading constant

$3n^3 + 2n^2 - 18n + 1028 \Rightarrow 3n^3 \Rightarrow n^3$

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

$2^{n/3+100} + 100n^{100} \Rightarrow 2^{n/3+100} \Rightarrow 2^{n/3}$
Asymptotic Analysis: $O$-notation

- Ignoring lower order terms
- Ignoring leading constant

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

- Ignoring lower order terms
- Ignoring leading constant

\( O \)-notation allows us to
- ignore architecture of computer
- ignore programming language
Asymptotic Analysis of Insertion Sort

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

1. for \(j \leftarrow 2\) to \(n\)
2. \(key \leftarrow A[j]\)
3. \(i \leftarrow j - 1\)
4. while \(i > 0\) and \(A[i] > key\)
   5. \(A[i + 1] \leftarrow A[i]\)
   6. \(i \leftarrow i - 1\)
7. \(A[i + 1] \leftarrow key\)

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

Total running time = \(\sum_{j=2}^{n} O(j) = O(n^2)\) (informal)
Asymptotic Analysis of Insertion Sort

insertion-sort(A, n)

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

- Worst-case running time for iteration $j$ in the outer loop?
Asymptotic Analysis of Insertion Sort

```
insertion-sort(A, n)

1. for j ← 2 to n
2.   key ← A[j]
3.   i ← j - 1
4.   while i > 0 and A[i] > key
6.     i ← i - 1
7.   A[i + 1] ← key
```

- Worst-case running time for iteration $j$ in the outer loop?
  Answer: $O(j)$
Asymptotic Analysis of Insertion Sort

insertion-sort($A, n$)

1. for $j \leftarrow 2$ to $n$
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j - 1$
4. while $i > 0$ and $A[i] > \text{key}$
   4.1. $A[i + 1] \leftarrow A[i]$
   4.2. $i \leftarrow i - 1$
5. $A[i + 1] \leftarrow \text{key}$

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

Total running time = $\sum_{j=2}^{n} O(j) = O(n^2)$ (informal)

Basic operations take $O(1)$ time: addition, subtraction, multiplication, etc.

Each integer (word) has $c \log n$ bits, $c \geq 1$ large enough

Basic operations take $O(1)$ time: addition, subtraction, multiplication, etc.

Each integer (word) has $c \log n$ bits, $c \geq 1$ large enough

Precision of real numbers?
Computation Model

- Basic operations take $O(1)$ time: addition, subtraction, multiplication, etc.
- Each integer (word) has $c \log n$ bits, $c \geq 1$ large enough
- Precision of real numbers? Try to avoid using real numbers in this course.
Computation Model

- Basic operations take $O(1)$ time: addition, subtraction, multiplication, etc.
- Each integer (word) has $c \log n$ bits, $c \geq 1$ large enough
- Precision of real numbers? Try to avoid using real numbers in this course.
- Can we do better than insertion sort asymptotically?

Basic operations take $O(1)$ time: addition, subtraction, multiplication, etc.

Each integer (word) has $c \log n$ bits, $c \geq 1$ large enough

Precision of real numbers?
Try to avoid using real numbers in this course.

Can we do better than insertion sort asymptotically?

Yes: merge sort, quicksort, heap sort, ...
Remember to sign up for Piazza.

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 $f(n) > 0$
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 $n$. 

- $2^n - n - 30$ is asymptotically positive.
- $2n - n^2 - 20$ is not asymptotically positive.
- $100n - n^2/10 + 50$ is not asymptotically positive.

We only consider 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 $n$.

- $n^2 - n - 30$
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 \( n \).

- \( n^2 - n - 30 \) \quad Yes
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 \( n \).

- \( n^2 - n - 30 \quad \text{Yes} \)

- \( 2^n - n^{20} \)
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
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 $n$.

- $n^2 - n - 30$ \quad Yes
- $2^n - n^{20}$ \quad Yes
- $100n - n^2/10 + 50$?
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 \( n \).
- \( n^2 - n - 30 \) Yes
- \( 2^n - n^{20} \) Yes
- \( 100n - n^2/10 + 50? \) No
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 $n$.

- $n^2 - n - 30$ Yes
- $2^n - n^{20}$ Yes
- $100n - n^2/10 + 50?$ No

We only consider asymptotically positive functions.
**O-Notation** For a function $g(n)$,

\[ O(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0 \}. \]
**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 other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c$ and large enough $n$. 
**O-Notation** For a function $g(n)$,

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

- In other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c$ and large enough $n$. 

![Graph showing O-Notation](image-url)
**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 other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c$ and large enough $n$.

- $3n^2 + 2n \in O(n^2 - 10n)$
**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 other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c$ and large enough $n$.

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

**Proof.**

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

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

$$= -n^2 + 40n \leq 0.$$ 

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

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

- In other words, \( f(n) \in O(g(n)) \) if \( f(n) \leq cg(n) \) for some \( c \) and large enough \( n \).

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

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

- In other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c$ and large enough $n$.

- $3n^2 + 2n \in O(n^2 - 10n)$
- $3n^2 + 2n \in O(n^3 - 5n^2)$
**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 other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c$ and large enough $n$.

- $3n^2 + 2n \in O(n^2 - 10n)$
- $3n^2 + 2n \in O(n^3 - 5n^2)$
- $n^{100} \in O(2^n)$
**O-Notation** For a function $g(n)$,

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

- In other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c$ and large enough $n$.

- $3n^2 + 2n \in O(n^2 - 10n)$
- $3n^2 + 2n \in O(n^3 - 5n^2)$
- $n^{100} \in O(2^n)$
- $n^3 \notin O(10n^2)$
**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 other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c$ and large enough $n$.

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

<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 “\( f(n) \in O(g(n)) \)”
We use “\( f(n) = O(g(n)) \)” to denote “\( f(n) \in O(g(n)) \)”

- \( 3n^2 + 2n = O(n^3 - 10n) \)
- \( 3n^2 + 2n = O(n^2 + 5n) \)
- \( 3n^2 + 2n = O(n^2) \)
We use “\( f(n) = O(g(n)) \)” to denote “\( f(n) \in O(g(n)) \)”

- \( 3n^2 + 2n = O(n^3 - 10n) \)
- \( 3n^2 + 2n = O(n^2 + 5n) \)
- \( 3n^2 + 2n = O(n^2) \)

“\( = \)” is asymmetric! Following statements are wrong:
Conventions

- We use \( f(n) = \mathcal{O}(g(n)) \) to denote \( f(n) \in \mathcal{O}(g(n)) \)
- \( 3n^2 + 2n = \mathcal{O}(n^3 - 10n) \)
- \( 3n^2 + 2n = \mathcal{O}(n^2 + 5n) \)
- \( 3n^2 + 2n = \mathcal{O}(n^2) \)

“\( = \)” is asymmetric! Following statements are wrong:
- \( \mathcal{O}(n^3 - 10n) = 3n^2 + 2n \)
- \( \mathcal{O}(n^2 + 5n) = 3n^2 + 2n \)
- \( \mathcal{O}(n^2) = 3n^2 + 2n \)
**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 $g(n)$,

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

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

- In other words, $f(n) \in Ω(g(n))$ if $f(n) \geq cg(n)$ for some $c$ and large enough $n$. 
\( \Omega \)-Notation: Asymptotic Lower Bound

For a function \( g(n) \),
\[
\Omega(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \geq cg(n), \forall n \geq n_0 \}\].
Again, we use “=” instead of \( \in \).

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

- \(4n^2 = \Omega(n - 10)\)
- \(3n^2 - n + 10 = \Omega(n^2 - 20)\)

<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>(\geq)</td>
<td></td>
</tr>
</tbody>
</table>
Again, we use “=” instead of $\in$.

- $4n^2 = \Omega(n - 10)$
- $3n^2 - n + 10 = \Omega(n^2 - 20)$

<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>$\geq$</td>
<td></td>
</tr>
</tbody>
</table>

**Theorem**  
$f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$. 
**Θ-Notation** For a function $g(n)$,

$$\Theta(g(n)) = \{ \text{function } f : \exists c_2 \geq c_1 > 0, n_0 > 0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}.$$
\textbf{Θ-Notation: Asymptotic Tight Bound}

\textbf{Θ-Notation} For a function $g(n)$,

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

\[ c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}. \]

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

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

- \( f(n) = \Theta(g(n)) \), then for large enough \( n \), we have “\( f(n) \approx g(n) \)”. 
**Θ-Notation** For a function \( g(n) \),

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

- \( 3n^2 + 2n = \Theta(n^2 - 20n) \)
**Θ-Notation: Asymptotic Tight Bound**

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

- \( 3n^2 + 2n = \Theta(n^2 - 20n) \)
- \( 2^{n/3} + 100 = \Theta(2^{n/3}) \)
**Θ-Notation: Asymptotic Tight Bound**

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

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

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

<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>$\geq$</td>
<td>$=$</td>
</tr>
</tbody>
</table>
**Θ-Notation**: For a function \( g(n) \),

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

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

<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>( \geq )</td>
<td>( = )</td>
</tr>
</tbody>
</table>

**Theorem** \( f(n) = \Theta(g(n)) \) if and only if \( f(n) = O(g(n)) \) and \( f(n) = \Omega(g(n)) \).
Exercise

For each pair of functions $f, g$ in the following table, indicate whether $f$ is $O$, $\Omega$ or $\Theta$ of $g$.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>$g$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n^3 - 100n$</td>
<td>$5n^2 + 3n$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$3n - 50$</td>
<td>$n^2 - 7n$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n^2 - 100n$</td>
<td>$5n^2 + 30n$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lg_{10} n$</td>
<td>$n^{0.1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2^n$</td>
<td>$2^{n/2}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sqrt{n}$</td>
<td>$n^{\sin n}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise

For each pair of functions $f, g$ in the following table, indicate whether $f$ is $O$, $\Omega$ or $\Theta$ of $g$.

<table>
<thead>
<tr>
<th>$f$</th>
<th>$g$</th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^3 - 100n$</td>
<td>$5n^2 + 3n$</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>$3n - 50$</td>
<td>$n^2 - 7n$</td>
<td>No</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>$n^2 - 100n$</td>
<td>$5n^2 + 30n$</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>$\log_{10} n$</td>
<td>$n^{0.1}$</td>
<td>No</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>$2^n$</td>
<td>$2^{n/2}$</td>
<td>No</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>$\sqrt{n}$</td>
<td>$n^{\sin n}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Exercise**

For each pair of functions $f, g$ in the following table, indicate whether $f$ is $O$, $\Omega$ or $\Theta$ of $g$.

<table>
<thead>
<tr>
<th>$f$</th>
<th>$g$</th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^3 - 100n$</td>
<td>$5n^2 + 3n$</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>$3n - 50$</td>
<td>$n^2 - 7n$</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>$n^2 - 100n$</td>
<td>$5n^2 + 30n$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lg_{10} n$</td>
<td>$n^{0.1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2^n$</td>
<td>$2^{n/2}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sqrt{n}$</td>
<td>$n^{\sin n}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise

For each pair of functions $f, g$ in the following table, indicate whether $f$ is $O$, $\Omega$ or $\Theta$ of $g$.

<table>
<thead>
<tr>
<th>$f$</th>
<th>$g$</th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^3 - 100n$</td>
<td>$5n^2 + 3n$</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>$3n - 50$</td>
<td>$n^2 - 7n$</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>$n^2 - 100n$</td>
<td>$5n^2 + 30n$</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$\log_{10} n$</td>
<td>$n^{0.1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2^n$</td>
<td>$2^{n/2}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sqrt{n}$</td>
<td>$n^{\sin n}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Exercise**

For each pair of functions $f, g$ in the following table, indicate whether $f$ is $O, \Omega$ or $\Theta$ of $g$.

<table>
<thead>
<tr>
<th>$f$</th>
<th>$g$</th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^3 - 100n$</td>
<td>$5n^2 + 3n$</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>$3n - 50$</td>
<td>$n^2 - 7n$</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>$n^2 - 100n$</td>
<td>$5n^2 + 30n$</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$\log_{10} n$</td>
<td>$n^{0.1}$</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>$2^n$</td>
<td>$2^{n/2}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sqrt{n}$</td>
<td>$n^{\sin n}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise

For each pair of functions $f, g$ in the following table, indicate whether $f$ is $O$, $Ω$ or $Θ$ of $g$.

<table>
<thead>
<tr>
<th>$f$</th>
<th>$g$</th>
<th>$O$</th>
<th>$Ω$</th>
<th>$Θ$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^3 - 100n$</td>
<td>$5n^2 + 3n$</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>$3n - 50$</td>
<td>$n^2 - 7n$</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>$n^2 - 100n$</td>
<td>$5n^2 + 30n$</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$\log_{10} n$</td>
<td>$n^{0.1}$</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>$2^n$</td>
<td>$2^{n/2}$</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>$\sqrt{n}$</td>
<td>$n^{\sin n}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise

For each pair of functions $f, g$ in the following table, indicate whether $f$ is $O$, $\Omega$ or $\Theta$ of $g$.

<table>
<thead>
<tr>
<th>$f$</th>
<th>$g$</th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^3 - 100n$</td>
<td>$5n^2 + 3n$</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>$3n - 50$</td>
<td>$n^2 - 7n$</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>$n^2 - 100n$</td>
<td>$5n^2 + 30n$</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$\log_{10} n$</td>
<td>$n^{0.1}$</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>$2^n$</td>
<td>$2^{n/2}$</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>$\sqrt{n}$</td>
<td>$n^{\sin n}$</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Asymptotic Notations</td>
<td>$O$</td>
<td>$\Omega$</td>
<td>$\Theta$</td>
<td></td>
</tr>
<tr>
<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></td>
</tr>
<tr>
<td>Asymptotic Notations</td>
<td>$O$</td>
<td>$\Omega$</td>
<td>$\Theta$</td>
<td></td>
</tr>
<tr>
<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></td>
</tr>
</tbody>
</table>

**Trivial Facts on Comparison Relations**

- $f \leq g \iff g \geq f$
- $f = g \iff f \leq g \text{ and } f \geq g$
- $f \leq g \text{ or } f \geq g$
Asymptotic Notations

<table>
<thead>
<tr>
<th></th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Relations</td>
<td>$\leq$</td>
<td>$\geq$</td>
<td>$=$</td>
</tr>
</tbody>
</table>

Trivial Facts on Comparison Relations

- $f \leq g \iff g \geq f$
- $f = g \iff f \leq g \text{ and } f \geq g$
- $f \leq g \text{ or } f \geq g$

Correct Analogies

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

<table>
<thead>
<tr>
<th></th>
<th>$O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Relations</td>
<td>$\leq$</td>
<td>$\geq$</td>
<td>$=$</td>
</tr>
</tbody>
</table>

Trivial Facts on Comparison Relations

- $f \leq g \iff g \geq f$
- $f = g \iff f \leq g$ and $f \geq g$
- $f \leq g$ or $f \geq g$

Correct Analogies

- $f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$
- $f(n) = \Theta(g(n)) \iff f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Incorrect Analogy

- $f(n) = O(g(n))$ or $g(n) = O(f(n))$
Incorrect Analogy

- \( f(n) = O(g(n)) \) or \( g(n) = O(f(n)) \)
Incorrect Analogy

- $f(n) = O(g(n))$ or $g(n) = O(f(n))$

\[
\begin{align*}
  f(n) &= n^2 \\
  g(n) &= \begin{cases} 
  1 & \text{if } n \text{ is odd} \\
  2^n & \text{if } n \text{ is even} 
  \end{cases}
\end{align*}
\]
Recall: informal way to define $O$-notation

- ignoring lower order terms: $3n^2 - 10n - 5 \rightarrow 3n^2$
- ignoring leading constant: $3n^2 \rightarrow n^2$
Recall: informal way to define $O$-notation

- ignoring lower order terms: $3n^2 - 10n - 5 \rightarrow 3n^2$
- ignoring leading constant: $3n^2 \rightarrow n^2$
- $3n^2 - 10n - 5 = O(n^2)$
Recall: informal way to define $O$-notation

- ignoring lower order terms: $3n^2 - 10n - 5 \to 3n^2$
- ignoring leading constant: $3n^2 \to n^2$
- $3n^2 - 10n - 5 = O(n^2)$
- Indeed, $3n^2 - 10n - 5 = \Omega(n^2), 3n^2 - 10n - 5 = \Theta(n^2)$
Recall: informal way to define $O$-notation

- ignoring lower order terms: $3n^2 - 10n - 5 \to 3n^2$
- ignoring leading constant: $3n^2 \to n^2$
- $3n^2 - 10n - 5 = O(n^2)$
- Indeed, $3n^2 - 10n - 5 = \Omega(n^2), 3n^2 - 10n - 5 = \Theta(n^2)$
- theoretically, nothing tells us to ignore lower order terms and leading constant.
Recall: informal way to define $O$-notation

- ignoring lower order terms: $3n^2 - 10n - 5 \rightarrow 3n^2$
- ignoring leading constant: $3n^2 \rightarrow n^2$
- $3n^2 - 10n - 5 = O(n^2)$
- Indeed, $3n^2 - 10n - 5 = \Omega(n^2), 3n^2 - 10n - 5 = \Theta(n^2)$
- theoretically, nothing tells us to ignore lower order terms and leading constant.
- $3n^2 - 10n - 5 = O(5n^2 - 6n + 5)$ is correct, though weird
Recall: informal way to define $O$-notation

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

- theoretically, 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 simpler.
Notice that $O$ denotes asymptotic upper bound.

- $n^2 + 2n = O(n^3)$ is correct.
- The following sentence is correct: the running time of the insertion sort algorithm is $O(n^4)$. 
Notice that \( O \) denotes asymptotic upper bound.

- \( n^2 + 2n = O(n^3) \) is correct.
- The following sentence is correct: the running time of the insertion sort algorithm is \( O(n^4) \).
- We do not use \( \Omega \) and \( \Theta \) very often when we talk about running times.
- We say: the running time of the insertion sort algorithm is \( O(n^2) \) and the bound is tight.
$o$ and $\omega$-Notations

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

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

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

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

Example:

- $3n^2 + 5n + 10 = o(n^3)$, but $3n^2 + 5n + 10 \neq o(n^2)$.
- $3n^2 + 5n + 10 = \omega(n)$, but $3n^2 + 5n + 10 \neq \omega(n^2)$. 
<table>
<thead>
<tr>
<th>Asymptotic Notations: $O$</th>
<th>$\Omega$</th>
<th>$\Theta$</th>
<th>$o$</th>
<th>$\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Relations: $\leq$</td>
<td>$\geq$</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>
</tr>
<tr>
<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>
</tr>
</tbody>
</table>

Questions?
Outline

1 Syllabus

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

\[ \text{sum}(A, n) \]

1. $S \leftarrow 0$
2. for $i \leftarrow 1$ to $n$
3. \hspace{1em} $S \leftarrow S + A[i]$
4. return $S$
$O(n)$ (Linear) Running Time

- Merge two sorted arrays

<table>
<thead>
<tr>
<th>3</th>
<th>8</th>
<th>12</th>
<th>20</th>
<th>32</th>
<th>48</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 two sorted arrays
Merge two sorted arrays

\begin{center}
\begin{tabular}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
5 & 7 & 9 & 25 & 29 & 3
\end{tabular}
\end{center}
Merge two sorted arrays
Merge two sorted arrays

\[
\begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
5 & 7 & 9 & 25 & 29 \\
3 & 5 \\
\end{array}
\]
$O(n)$ (Linear) Running Time

- Merge two sorted arrays

```
3 8 12 20 32 48
5 7 9 25 29
3 5
```
Merge two sorted arrays
\( O(n) \) (Linear) Running Time

- Merge two sorted arrays

\[
\begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
5 & 7 & 9 & 25 & 29 \\
3 & 5 & 7 \\
\end{array}
\]
**O(n) (Linear) Running Time**

- Merge two sorted arrays

\[
\begin{array}{cccccc}
3 & 8 & 12 & 20 & 32 & 48 \\
5 & 7 & 9 & 25 & 29 \\
3 & 5 & 7 & 8 \\
\end{array}
\]
Merge two sorted arrays
\( O(n) \) (Linear) Running Time

- Merge two sorted arrays

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

\begin{align*}
3 & \quad 8 & \quad 12 & \quad 20 & \quad 32 & \quad 48 \\
5 & \quad 7 & \quad 9 & \quad 25 & \quad 29 & \\
3 & \quad 5 & \quad 7 & \quad 8 & \quad 9 & \quad 12 & \quad 20 & \quad 25 & \quad 29 & \quad 32 & \quad 48
\end{align*}
**O(n) (Linear) Running Time**

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

1. \( A \leftarrow [] \); \( i \leftarrow 1 \); \( j \leftarrow 1 \)
2. while \( i \leq n_1 \) and \( j \leq n_2 \)
3. \quad if \( (B[i] \leq C[j]) \) then
4. \quad \quad append \( B[i] \) to \( A \); \( i \leftarrow i + 1 \)
5. \quad else
6. \quad \quad append \( C[j] \) to \( A \); \( j \leftarrow j + 1 \)
7. \quad if \( i \leq n_1 \) then append \( B[i..n_1] \) to \( A \)
8. \quad 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($B, C, n_1, n_2$) \ \ \ \ $B$ and $C$ are sorted, with length $n_1$ and $n_2$

1. $A \leftarrow []; i \leftarrow 1; j \leftarrow 1$
2. while $i \leq n_1$ and $j \leq n_2$
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. else
4. $B \leftarrow \text{merge-sort}\left(A[1..\lfloor n/2 \rfloor], \lceil n/2 \rceil \right)$
5. $C \leftarrow \text{merge-sort}\left(A[\lfloor n/2 \rfloor + 1..n], n - \lfloor n/2 \rfloor \right)$
6. return $\text{merge}(B, C, \lfloor n/2 \rfloor, n - \lfloor n/2 \rfloor)$
Merge-Sort

- Each level takes running time $O(n)$.
- There are $O(\lg n)$ levels.
- Running time is $O(n \lg n)$. 
**Merge-Sort**

![Tree Diagram]

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

Running time = $O(n \log n)$
$O(n \lg n)$ Running Time

- **Merge-Sort**

  ![Merge-Sort Diagram]

  - Each level takes running time $O(n)$
  - There are $O(\lg n)$ levels
**$O(n \lg n)$ Running Time**

- **Merge-Sort**

![Diagram](attachment:image.png)

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

\( O(n^2) \) (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

```plaintext
closest-pair(x, y, n)
1 bestd ← ∞
2 for i ← 1 to $n - 1$
3   for j ← $i + 1$ to $n$
4     $d ← \sqrt{(x[i] - x[j])^2 + (y[i] - y[j])^2}$
5     if $d < bestd$ then
6       besti ← i, bestj ← j, bestd ← $d$
7 return $(besti, bestj)$
```
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($x, y, n$)

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

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

\begin{verbatim}
matrix-multiplication(A, B, n)
1. $C \leftarrow$ matrix of size $n \times n$, with all entries being 0
2. for $i \leftarrow 1$ to $n$
3.   for $j \leftarrow 1$ to $n$
4.     for $k \leftarrow 1$ to $n$
5.       $C[i, k] \leftarrow C[i, k] + A[i, j] \times B[j, k]$
6. return $C$
\end{verbatim}
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 $(u, v) \notin E$. 
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 $(u, v) \notin E$. 

Def.
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 $(u, v) \notin E$. 

$O(n^k)$ Running Time for Integer $k \geq 4$
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 $(u, v) \notin E$.

Independent set of size $k$

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

**Output:** whether there is an independent set of size $k$
Independent Set of Size $k$

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

**Output:** whether there is an independent set of size $k$

```plaintext
independent-set(G = (V, E))

1. for every set $S \subseteq V$ of size $k$
2. \hspace{1em} $b \leftarrow$ true
3. \hspace{1em} for every $u, v \in S$
4. \hspace{2em} if $(u, v) \in E$ then $b \leftarrow$ false
5. \hspace{1em} if $b$ return true
6. return false
```

Running time $= \mathcal{O}(\frac{n^k}{k!} \times k^2) = \mathcal{O}(n^k)$ (assume $k$ is a constant)
Beyond Polynomial Time: $O(2^n)$

**Maximum Independent Set Problem**

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

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

```plaintext
max-independent-set(\(G = (V, E)\))

1. \(R \leftarrow \emptyset\)
2. for every set \(S \subseteq V\)
3. \(b \leftarrow \text{true}\)
4. for every \(u, v \in S\)
5. \(\text{if } (u, v) \in E \text{ then } b \leftarrow \text{false}\)
6. \(\text{if } b \text{ and } |S| > |R| \text{ then } R \leftarrow S\)
7. return \(R\)
```

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

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: $O(n!)$

**Hamiltonian Cycle Problem**

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

**Output:** a cycle that visits each node exactly once, or say no such cycle exists
Hamiltonian\((G = (V, E))\)

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

Running time = \(O(n! \times n)\)
$O(\lg n)$ (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:
$O(\lg n)$ (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:
$O(\log n)$ (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
```
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:
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:
**$O(\lg n)$ (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
```

42 > 35
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:
$O(\lg n)$ (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
```
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:

<table>
<thead>
<tr>
<th>3</th>
<th>8</th>
<th>10</th>
<th>25</th>
<th>29</th>
<th>37</th>
<th>38</th>
<th>42</th>
<th>46</th>
<th>52</th>
<th>59</th>
<th>61</th>
<th>63</th>
<th>75</th>
<th>79</th>
</tr>
</thead>
</table>

$25 < 35$
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:
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:
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:
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:

$$
\begin{array}{cccccccccccc}
3 & 8 & 10 & 25 & 29 & 37 & 38 & 42 & 46 & 52 & 59 & 61 & 63 & 75 & 79 \\
\end{array}
$$
Binary search

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

**binary-search($A$, $n$, $t$)**

1. $i \leftarrow 1$, $j \leftarrow n$
2. while $i \leq j$ do
3.     $k \leftarrow \lfloor (i + j)/2 \rfloor$
4.     if $A[k] = t$ return true
5.     if $A[k] < t$ then $j \leftarrow k - 1$ else $i \leftarrow k + 1$
6. return false

$O(\log n)$ (Logarithmic) Running Time
**O(\(\lg n\)) (Logarithmic) Running Time**

Binary search

- Input: sorted array \(A\) of size \(n\), an integer \(t\);
- Output: whether \(t\) appears in \(A\).

```plaintext
binary-search(\(A, n, t\))

1. \(i \leftarrow 1, j \leftarrow n\)
2. while \(i \leq j\) do
3. \(k \leftarrow \lfloor (i + j)/2 \rfloor\)
4. if \(A[k] = t\) return true
5. if \(A[k] < t\) then \(j \leftarrow k - 1\) else \(i \leftarrow k + 1\)
6. return false
```

Running time = \(O(\lg n)\)
Compare the Orders

- Sort the functions from smallest to largest asymptotically
  \( n^{\sqrt{n}}, \ lg\ n, \ n, \ n^2, \ n\ lg\ n, \ n!, \ 2^n, \ e^n, \ lg(n!), \ n^n \)
Compare the Orders

- Sort the functions from smallest to largest asymptotically
  \( n^{\sqrt{n}}, \ lg\ n, \ n, \ n^2, \ n\ lg\ n, \ n!, \ 2^n, \ e^n, \ lg(n!), \ n^n \)
- \( f < g \) stands for \( f = o(g) \), \( f = g \) stands for \( f = \Theta(g) \)
- \( lg\ n < n^{\sqrt{n}} \)
Comparing the Orders

- Sort the functions from smallest to largest asymptotically:
  \[ n^{\sqrt{n}}, \lg n, n, n^2, n \lg n, n!, 2^n, e^n, \lg(n!), n^n \]
- \( f < g \) stands for \( f = o(g) \), \( f = g \) stands for \( f = \Theta(g) \)!
- \( \lg n < n^{\sqrt{n}} \)
- \( \lg n < n < n^{\sqrt{n}} \)
Compare the Orders

- Sort the functions from smallest to largest asymptotically
  \( n^{\sqrt{n}}, \lg n, n, n^2, n \lg n, n!, 2^n, e^n, \lg(n!), n^n \)
- \( f < g \) stands for \( f = o(g) \), \( f = g \) stands for \( f = \Theta(g) \)
- \( \lg n < n^{\sqrt{n}} \)
- \( \lg n < n < n^{\sqrt{n}} \)
- \( \lg n < n < n^2 < n^{\sqrt{n}} \)
Compare the Orders

- Sort the functions from smallest to largest asymptotically
  \( n^{\sqrt{n}}, \ lg\ n, \ n, \ n^2, \ n\ lg\ n, \ n!, \ 2^n, \ e^n, \ lg(n!), \ n^n \)
- \( f < g \) stands for \( f = o(g) \), \( f = g \) stands for \( f = \Theta(g) \)!
- \( lg\ n < n^{\sqrt{n}} \)
- \( lg\ n < n < n^{\sqrt{n}} \)
- \( lg\ n < n < n^2 < n^{\sqrt{n}} \)
- \( lg\ n < n < n\ lg\ n < n^2 < n^{\sqrt{n}} \)
Compare the Orders

- Sort the functions from smallest to largest asymptotically:
  \[ n^{\sqrt{n}}, \ lg\ n, \ n, \ n^2, \ n \ lg\ n, \ n!, \ 2^n, \ e^n, \ lg(n!), \ n^n \]
- \( f < g \) stands for \( f = o(g) \), \( f = g \) stands for \( f = \Theta(g) \)
- \( \lg\ n < n^{\sqrt{n}} \)
- \( \lg\ n < n < n^{\sqrt{n}} \)
- \( \lg\ n < n < n^2 < n^{\sqrt{n}} \)
- \( \lg\ n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} \)
- \( \lg\ n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} < n! \)
Compare the Orders

- Sort the functions from smallest to largest asymptotically
  \( n^{\sqrt{n}}, \ lg\ n, \ n, \ n^2, \ n\ lg\ n, \ n!, \ 2^n, \ e^n, \ lg(n!), \ n^n \)
- \( f < g \) stands for \( f = o(g) \), \( f = g \) stands for \( f = \Theta(g) \)
- \( \lg n < n^{\sqrt{n}} \)
- \( \lg n < n < n^{\sqrt{n}} \)
- \( \lg n < n < n^2 < n^{\sqrt{n}} \)
- \( \lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} \)
- \( \lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} < n! \)
- \( \lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} < 2^n < n! \)
Compare the Orders

- Sort the functions from smallest to largest asymptotically:
  \[ n^{\sqrt{n}}, \ lg\ n, \ n, \ n^2, \ n\ lg\ n, \ n!, \ 2^n, \ e^n, \ lg(n!), \ n^n \]
- \( f < g \) stands for \( f = o(g) \), \( f = g \) stands for \( f = \Theta(g) \)!
- \( \lg n < n^{\sqrt{n}} \)
- \( \lg n < n < n^{\sqrt{n}} \)
- \( \lg n < n < n^2 < n^{\sqrt{n}} \)
- \( \lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} \)
- \( \lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} < n! \)
- \( \lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} < 2^n < n! \)
- \( \lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} < 2^n < e^n < n! \)
Compare the Orders

- Sort the functions from smallest to largest asymptotically:
  \(n^{\sqrt{n}}, \ lg\ n, \ n, \ n^2, \ n \ lg\ n, \ n!, \ 2^n, \ e^n, \ lg(n!), \ n^n\)

- \(f < g\) stands for \(f = o(g)\), \(f = g\) stands for \(f = \Theta(g)\)

- \(\lg n < n^{\sqrt{n}}\)
- \(\lg n < n < n^{\sqrt{n}}\)
- \(\lg n < n < n^2 < n^{\sqrt{n}}\)
- \(\lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}}\)
- \(\lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} < n!\)
- \(\lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} < 2^n < n!\)
- \(\lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} < 2^n < e^n < n!\)
- \(\lg n < n < n \ lg\ n = \lg(n!) < n^2 < n^{\sqrt{n}} < 2^n < e^n < n!\)
Compare the Orders

- Sort the functions from smallest to largest asymptotically
  \[ n^{\sqrt{n}}, \ lg\ n, \ n, \ n^2, \ n \ lg\ n, \ n!, \ 2^n, \ e^n, \ lg(n!), \ n^n \]
- \( f < g \) stands for \( f = o(g) \), \( f = g \) stands for \( f = \Theta(g) \)

- \( \lg n < n^{\sqrt{n}} \)
- \( \lg n < n < n^{\sqrt{n}} \)
- \( \lg n < n < n^2 < n^{\sqrt{n}} \)
- \( \lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} \)
- \( \lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} < n! \)
- \( \lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} < 2^n < n! \)
- \( \lg n < n < n \ lg\ n < n^2 < n^{\sqrt{n}} < 2^n < e^n < n! \)
- \( \lg n < n < n \ lg\ n = \ lg(n!) < n^2 < n^{\sqrt{n}} < 2^n < e^n < n! \)
- \( \lg n < n < n \ lg\ n = \ lg(n!) < n^2 < n^{\sqrt{n}} < 2^n < e^n < n! < n^n \)
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: $o(n^2)$
Goal of Algorithm Design

- Design algorithms to minimize the order of the running time.
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.
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: Does ignoring the leading constant cause any issues?

- e.g., how can we compare an algorithm with running time $0.1n^2$ with an algorithm with running time $1000n$?
Q: Does ignoring the leading constant cause any issues?

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

A:
Q: Does ignoring the leading constant cause any issues?

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

A:
- Sometimes yes
Q: Does ignoring the leading constant cause any issues?

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

A:
- Sometimes yes
- However, when $n$ is big enough, $1000n < 0.1n^2$
Q: Does ignoring the leading constant cause any issues?

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

A:
- Sometimes yes
- However, when $n$ is big enough, $1000n < 0.1n^2$
- For “natural” algorithms, constants are not so big!
Q: Does ignoring the leading constant cause any issues?

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

A:

- Sometimes yes
- However, when $n$ is big enough, $1000n < 0.1n^2$
- For “natural” algorithms, constants are not so big!
- So, for reasonable $n$, algorithm with lower order running time beats algorithm with higher order running time.