CSE 431/531: Algorithm Analysis and Design (Spring 2022)

Introduction and Syllabus

Lecturer: Shi Li
Department of Computer Science and Engineering
University at Buffalo
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, and slides):  
http://www.cse.buffalo.edu/~shil/courses/CSE531/

Please sign up course on Piazza via link on course webpage  
- homeworks, solutions, announcements, polls, asking/answering questions
CSE 431/531: Algorithm Analysis and Design

- Time & Location: 9:00am-9:50am, NSC 201
- Instructor:
  - Shi Li, shil@buffalo.edu
- TAs and Graders:
  - Sean Sanders, Xiaoyu Zhang,
  - Graders: TBD
You **should** already have/know:

- **Mathematical Background**
  - basic reasoning skills, inductive proofs
- **Basic data Structures**
  - linked lists, arrays
  - stacks, queues
- **Some Programming Experience**
  - Python, C, C++ or Java
You Will Learn

- Classic algorithms for classic problems
  - Sorting, shortest paths, minimum spanning tree, ···
- How to analyze algorithms
  - Correctness
  - Running time (efficiency)
- Meta techniques to design algorithms
  - Greedy algorithms
  - Divide and conquer
  - Dynamic programming
  - ···
- NP-completeness
# Tentative Schedule

- 50 Minutes/Lecture × 42 Lectures

<table>
<thead>
<tr>
<th>Topic</th>
<th>Lectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>4 lectures</td>
</tr>
<tr>
<td>Graph Basics</td>
<td>3 lectures</td>
</tr>
<tr>
<td>Greedy Algorithms</td>
<td>7 lectures</td>
</tr>
<tr>
<td>Divide and Conquer</td>
<td>7 lectures</td>
</tr>
<tr>
<td>Dynamic Programming</td>
<td>7 lectures</td>
</tr>
<tr>
<td>Graph Algorithms</td>
<td>7 lectures</td>
</tr>
<tr>
<td>NP-Completeness</td>
<td>5 lectures</td>
</tr>
<tr>
<td>Final Review</td>
<td>2 lectures</td>
</tr>
</tbody>
</table>
Textbook

Textbook (Highly Recommended):

- **Algorithm Design**, 1st Edition, by *Jon Kleinberg* and *Eva Tardos*

Other Reference Books

Reading Before Classes

- Highly recommended: read the correspondent sections from the textbook (or reference book) before classes.
  - Sections for each lecture can be found on the course webpage.

- Slides are posted on course webpage. They may get updated before the classes start.

- In last lecture of a major topic (Greedy Algorithms, Divide and Conquer, Dynamic Programming, Graph Algorithms), I will discuss exercise problems, which will be posted on the course webpage before class.
Grading

- 40% for theory homeworks
  - 8 points × 5 theory homeworks
- 20% for programming problems
  - 10 points × 2 programming assignments
- 40% for final exam
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 submit an assignment solution for three days
- With no special reasons, no other late submissions will be accepted
Final Exam will be closed-book

**Academic Integrity (AI) Policy for the Course**

- **minor violation:**
  - 0 score for the involved homework/prog. assignment, **and**
  - 1-letter grade down
- **2 minor violations = 1 major violation**
  - failure for the course
  - case will be reported to the department and university
  - further sanctions may include a dishonesty mark on transcript or expulsion from university

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
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

- $\text{gcd}(270, 210) = \text{gcd}(210, 270 \mod 210) = \text{gcd}(210, 60)$
- $(270, 210) \rightarrow (210, 60) \rightarrow (60, 30) \rightarrow (30, 0)$
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 \(a'_1 \leq a'_2 \leq \cdots \leq a'_n\)

#### Example:

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

- Algorithms: insertion sort, merge sort, quicksort, ...
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: \hspace{1em} ($a, b$) ← ($b, a \mod b$)
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: \(A[i + 1] \leftarrow key\)

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

<table>
<thead>
<tr>
<th>12</th>
<th>15</th>
<th>21</th>
<th>35</th>
<th>53</th>
<th>59</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\uparrow)</td>
<td>(i)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
**Correctness of Insertion Sort**


  - 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 $n =$ worst running time over all possible arrays of length $n$
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
Asymptotic Analysis: $O$-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: for $j ← 2$ to $n$ do  
2:  
3: \hspace{1em} key ← $A[j]$  
4: \hspace{1em} $i ← j - 1$  
5: \hspace{1em} while $i > 0$ and $A[i] > key$ do  
6: \hspace{2em} $A[i + 1] ← A[i]$  
7: \hspace{2em} $i ← i - 1$  
8: \hspace{1em} $A[i + 1] ← key$

- **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
Remember to sign up for Piazza.

Questions?
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 other words, \( f(n) \in O(g(n)) \) if \( f(n) \leq cg(n) \) for some \( c > 0 \) and every large enough \( n \).
**$O$-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 $c > 0$ and every 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,

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

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

$$3n^2 + 2n \leq c(n^2 - 10n)$$

\[\square\]
**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 “\( f(n) \in O(g(n)) \)”
- \( 3n^2 + 2n = O(n^3 - 10n) \)
- \( 3n^2 + 2n = O(n^2 + 5n) \)
- \( 3n^2 + 2n = O(n^2) \)

“=” is asymmetric! Following equalities are wrong:

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

- Analogy: Mike is a student. A student is Mike.
**Ω-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 $n$.  

Ω-Notation: Asymptotic Lower Bound

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

$$\Omega(g(n)) = \{ \text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \geq cg(n), \forall n \geq n_0 \}.$$
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** \[ f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n)). \]
**Θ-Notation: Asymptotic Tight Bound**

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

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

$$c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \right\}.$$

- $f(n) = \Theta(g(n))$, then for large enough $n$, we have “$f(n) \approx g(n)$”.

![Diagram showing functions $f(n)$, $c_1 g(n)$, and $c_2 g(n)$ with a vertical dashed line at $n_0$.](image)
**Θ-Notation: Asymptotic Tight Bound**

**Θ-Notation** For a function $g(n)$,

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

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

<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 $f(n) = \Omega(g(n))$. 
### Asymptotic Notations

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

### Trivial Facts on Comparison Relations

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

### 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 $f(n) = \Omega(g(n))$
Incorrect Analogy

- \( f(n) = O(g(n)) \) or \( f(n) = \Omega(g(n)) \)

\[
\begin{align*}
  f(n) &= n^2 \\
  g(n) &= \begin{cases} 
    1 & \text{if } n \text{ is odd} \\
    n^3 & \text{if } n \text{ is even}
  \end{cases}
\end{align*}
\]
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, $3n^2 - 10n - 5 = \Omega(n^2), 3n^2 - 10n - 5 = \Theta(n^2)$

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 the insertion sort algorithm is $O(n^4)$.
- We say: the running time of the insertion sort algorithm is $O(n^2)$ and the bound is tight.
- We do not use $\Omega$ and $\Theta$ very often when we upper bound running times.
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>$n^3 - 100n$</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>$\log_2 n$</td>
<td>$\log_{10} n$</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>
</tbody>
</table>

We often use $\log n$ for $\log_2 n$. But for $O(\log n)$, the base is not important.
<table>
<thead>
<tr>
<th>Asymptotic Notations</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>

Questions?
Outline

1 Syllabus

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

3 Asymptotic Notations

4 Common Running times
$O(n)$ (Linear) Running Time

Computing the sum of $n$ numbers

**sum($A$, $n$)**

1. $S \leftarrow 0$
2. for $i \leftarrow 1$ to $n$
3. $S \leftarrow S + A[i]$
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>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>
```
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 $j \leq n_2$ do
3:     if $B[i] \leq C[j]$ then
4:         append $B[i]$ to $A$; $i \leftarrow i + 1$
5:     else
6:         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($A[1..\lfloor n/2 \rfloor], \lfloor n/2 \rfloor$)
4. $C \leftarrow$ merge-sort($A[\lfloor n/2 \rfloor + 1..n], n - \lfloor n/2 \rfloor$)
5. return merge($B, C, \lfloor n/2 \rfloor, n - \lfloor n/2 \rfloor$)
\( O(n \log n) \) Running Time

- Merge-Sort

\[ A[1..8] \]

\[ A[1..4] \quad A[5..8] \]


- Each level takes running time \( O(n) \)
- There are \( O(\log n) \) levels
- Running time = \( O(n \log n) \)
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: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$

**Output:** the pair of points that are closest

`closest-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 $n \times n$

\textbf{matrix-multiplication}(A, B, n)

1: $C \leftarrow$ matrix of size $n \times n$, with all entries being 0
2: \textbf{for} $i \leftarrow 1$ to $n$ \textbf{do}
3: \hspace{.5cm} \textbf{for} $j \leftarrow 1$ to $n$ \textbf{do}
4: \hspace{1cm} \textbf{for} $k \leftarrow 1$ to $n$ \textbf{do}
5: \hspace{1.5cm} $C[i, k] \leftarrow C[i, k] + A[i, j] \times B[j, k]$
6: \textbf{return} $C$
Beyond Polynomial Time: $2^n$

**Maximum Independent Set Problem**

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

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

**Algorithm:**

```plaintext
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{2em} 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: \hspace{1em} return “No Hamiltonian Cycle”

Running time = $O(n! \times n)$
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 $A$.

**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 $t < A[k]$ then $j \leftarrow k - 1$ else $i \leftarrow k + 1$
6. return false

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

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

- \( \log n = O(n) \)
- \( n = O(n^2) n = O(n \log n) \)
- \( n \log n = O(n^2) \)
- \( n^2 = O(n!) n^2 = O(2^n) \)
- \( 2^n = O(n!) 2^n = O(e^n) \)
- \( e^n = O(n!) \)
- \( n! = O(n^n) \)
Terminologies

When we talk about upper bound on running time:

- Logarithmic time: $O(\log n)$
- Linear time: $O(n)$
- Quadratic time $O(n^2)$
- Cubic time $O(n^3)$
- Polynomial time: $O(n^k)$ for some constant $k$
  - $O(n \log n) \subseteq O(n^{1.1})$. So, an $O(n \log n)$-time algorithm is also a polynomial time algorithm.
- 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.

- 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 $1000n$?

A:

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