# Examples

## Shortest Path

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

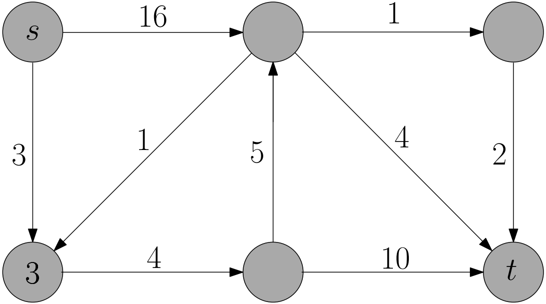
**Output:** a shortest path from $s$ to $t$ in $G$

# Examples

## Shortest Path

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

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

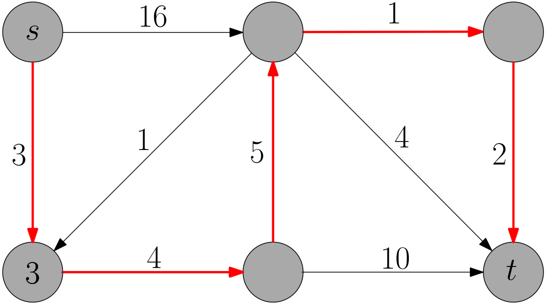


# Examples

## Shortest Path

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

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

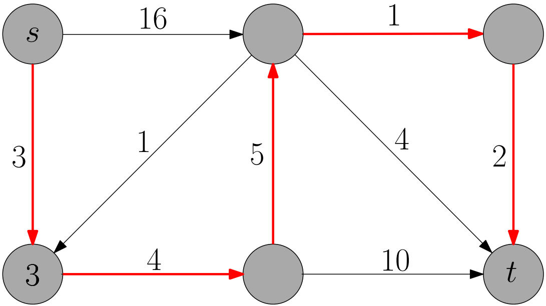


# Examples

## Shortest Path

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

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



- Algorithm: Dijkstra's algorithm . . .

# Examples

## Shortest Path

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

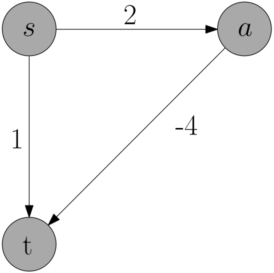
**Output:** a shortest path from $s$ to $t$ in $G$

# Examples

## Shortest Path

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

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

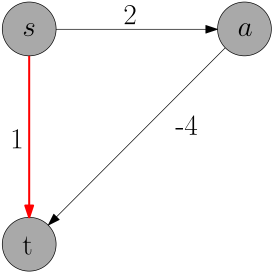


# Examples

## Shortest Path

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

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

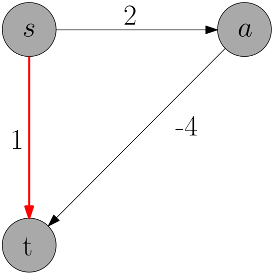


# Examples

## Shortest Path

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

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



- Algorithm: Bellman-Ford algorithm, Floyd-Warshall . . .

# 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

Pseudo-Code:

**Euclidean$(a, b)$**

```
while $b > 0$ do
    $(a, b) \leftarrow (b, a \bmod b)$
return $a$
```

Python program:

```
def euclidean(a: int, b: int):
    c = 0
    while b > 0:
        c = b
        b = a % b
        a = c
    return a
```

# Theoretical Analysis of Algorithms

- 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)
    - ...

# 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?

# 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

# 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)

# 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

# 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$(A, n)$

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

## Example:

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

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

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

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

| | | | | | |
|---|---|---|---|---|---|
| 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$(A, n)$**

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

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

$$\begin{array}{cccccc} 12 & 21 & 35 & 53 & 59 & 59 \\ & & & & \uparrow & \\ & & & & i & \end{array}$$

### Example:

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

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

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

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

$$\begin{array}{cccccc} 12 & 21 & 35 & 53 & 59 & 59 \\ & & & \uparrow & & \\ & & & i & & \end{array}$$

## Example:

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

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

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

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

12 21 35 53 53 59

↑

$i$

### Example:

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

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

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

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

$$12 \quad 21 \quad \underset{\underset{i}{\uparrow}}{35} \quad 53 \quad 53 \quad 59$$

### Example:

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

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

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

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

12 21 35 35 53 59

↑

$i$

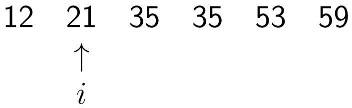
## Example:

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

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

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

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



12 21 35 35 53 59

$\uparrow$

$i$

**Example:**

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

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

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

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

12 21 21 35 53 59

$\uparrow$

$i$

## Example:

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

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

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

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

12 21 21 35 53 59

$\uparrow$

$i$

## Example:

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

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

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

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

$$\underset{\overset{\uparrow}{i}}{12} \quad 15 \quad 21 \quad 35 \quad 53 \quad 59$$

# 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

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

$$\text{after } j = 1 : 53, 12, 35, 21, 59, 15$$

$$\text{after } j = 2 : 12, 53, 35, 21, 59, 15$$

$$\text{after } j = 3 : 12, 35, 53, 21, 59, 15$$

$$\text{after } j = 4 : 12, 21, 35, 53, 59, 15$$

$$\text{after } j = 5 : 12, 21, 35, 53, 59, 15$$

$$\text{after } j = 6 : 12, 15, 21, 35, 53, 59$$

# Analyzing Running Time of Insertion Sort

- Q1: what is the size of input?

# Analyzing Running Time of Insertion Sort

- Q1: what is the size of input?
- A1: Running time as the function of size

# 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

# 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.

# 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 $n$

# Analyzing Running Time of Insertion Sort

- Q3: How fast is the computer?
- Q4: Programming language?

# Analyzing Running Time of Insertion Sort

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

# 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

# Asymptotic Analysis: $O$-notation

Informal way to define $O$-notation:

- Ignoring lower order terms
- Ignoring leading constant

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

# Asymptotic Analysis: $O$-notation

Informal way to define $O$-notation:

- Ignoring lower order terms
- Ignoring leading constant

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

# Asymptotic Analysis: $O$-notation

Informal way to define $O$-notation:

- Ignoring lower order terms
- Ignoring leading constant

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

- $n^2/100 - 3n + 10 \Rightarrow n^2/100 \Rightarrow n^2$

# 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

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

# Asymptotic Analysis: $O$-notation

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

$O$-notation allows us to ignore

- architecture of computer
- programming language
- how we measure the running time: seconds or # instructions?

# Asymptotic Analysis: $O$-notation

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

$O$-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

# Asymptotic Analysis: $O$-notation

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

$O$-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 $O(n)$

# 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 $n$!

# 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.

# 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 $4$

# 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$(A, n)$**

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

# Asymptotic Analysis of Insertion Sort

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

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

- Worst-case running time for iteration $j$ of the outer loop?

# Asymptotic Analysis of Insertion Sort

```
insertion-sort$(A, n)$
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

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

# Asymptotic Analysis of Insertion Sort

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

```
for $j \leftarrow 2$ to $n$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$
```

- Worst-case running time for iteration $j$ of the outer loop?
  Answer: $O(j)$
- Total running time $= \sum_{j=2}^{n} O(j) = O(\sum_{j=2}^{n} j)$
  $= O(\frac{n(n+1)}{2} - 1) = O(n^2)$

# Computation Model

# Computation Model

- Random-Access Machine (RAM) model
  - reading and writing $A[j]$ takes $O(1)$ time

# 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

# 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.

# 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?

# 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.

# 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?

# 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

# 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.** $f : \mathbb{N} \to \mathbb{R}$ 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 $n$.

# Asymptotically Positive Functions

**Def.** $f : \mathbb{N} \to \mathbb{R}$ 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 $n$.
- $n^2 - n - 30$

# Asymptotically Positive Functions

**Def.** $f : \mathbb{N} \to \mathbb{R}$ 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

# Asymptotically Positive Functions

**Def.** $f : \mathbb{N} \to \mathbb{R}$ 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
- $2^n - n^{20}$

# Asymptotically Positive Functions

**Def.** $f : \mathbb{N} \to \mathbb{R}$ 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

# Asymptotically Positive Functions

**Def.** $f : \mathbb{N} \to \mathbb{R}$ 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
- $100n - n^2/10 + 50$?

# Asymptotically Positive Functions

**Def.** $f : \mathbb{N} \to \mathbb{R}$ 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

# Asymptotically Positive Functions

**Def.** $f : \mathbb{N} \to \mathbb{R}$ 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: Asymptotic Upper Bound

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

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

# $O$-Notation: Asymptotic Upper Bound

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

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

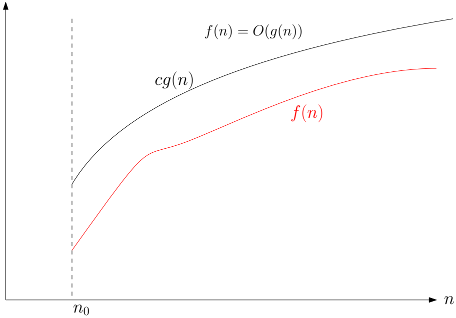
- In other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c > 0$ and every large enough $n$.

# $O$-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 $n$.



# $O$-Notation: Asymptotic Upper Bound

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

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

- In other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some $c > 0$ and every large enough $n$.
- $3n^2 + 2n \in O(n^2 - 10n)$

# $O$-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,

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

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

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

□

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

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

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

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

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

- 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 $g(n)$,

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

- In other words, $f(n) \in O(g(n))$ if $f(n) \le cg(n)$ 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 $n$.
- $3n^2 + 2n \in O(n^2 - 10n)$
- $3n^2 + 2n \in O(n^3 - 5n^2)$
- ? $n^{100} \in O(2^n)$
- ? $\sin n \in O(1/2)$

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

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

- In other words, $f(n) \in O(g(n))$ if $f(n) \le cg(n)$ for some $c$ and large enough $n$.
- $3n^2 + 2n \in O(n^2 - 10n)$
- $3n^2 + 2n \in O(n^3 - 5n^2)$
- ? $n^{100} \in O(2^n)$
- ? $\sin n \in O(1/2)$
- ? $n^3 \notin O(10n^2)$

**$O$-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 $n$.
- $3n^2 + 2n \in O(n^2 - 10n)$
- $3n^2 + 2n \in O(n^3 - 5n^2)$
- ? $n^{100} \in O(2^n)$
- ? $\sin n \in O(1/2)$
- ? $n^3 \notin O(10n^2)$

| Asymptotic Notations | $O$ | $\Omega$ | $\Theta$ |
|---|---|---|---|
| Comparison Relations | $\leq$ | | |