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

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

- 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 $g(n)$,

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

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

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

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

- 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 $g(n)$,

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

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

Asymptotic Notations	O	Ω	Θ
Comparison Relations	$\le$		

Conventions

- We use "$f(n) = O(g(n))$" to denote "$f(n) \in O(g(n))$"

Conventions

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

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:

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

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: 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 $g(n)$,

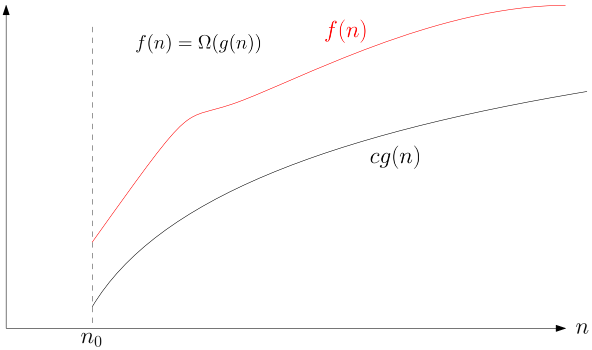
$$\Omega(g(n)) = \{\text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \geq cg(n), \forall n \geq n_0\}.$$

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

Ω-Notation: Asymptotic Lower Bound

Ω-Notation For a function $g(n)$,

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



Ω-Notation: Asymptotic Lower Bound

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

Ω-Notation: Asymptotic Lower Bound

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

Asymptotic Notations	O	Ω	Θ
Comparison Relations	$\leq$	$\geq$	

Ω-Notation: Asymptotic Lower Bound

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

Asymptotic Notations	O	Ω	Θ
Comparison Relations	$\leq$	$\geq$	

Theorem $f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$.

Θ-Notation: Asymptotic Tight Bound

Θ-Notation For a function $g(n)$,

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

Θ-Notation: Asymptotic Tight Bound

Θ-Notation For a function $g(n)$,

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

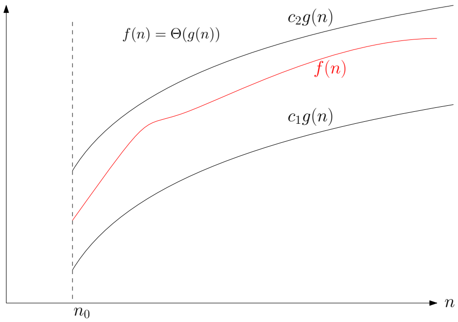
- $f(n) = \Theta(g(n))$, then for large enough n, we have "$f(n) \approx g(n)$".

Θ-Notation: Asymptotic Tight Bound

Θ-Notation For a function $g(n)$,

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

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



Θ-Notation: Asymptotic Tight Bound

Θ-Notation For a function $g(n)$,

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

- $3n^2 + 2n = \Theta(n^2 - 20n)$

Θ-Notation: Asymptotic Tight Bound

Θ-Notation For a function $g(n)$,

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

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

Θ-Notation: Asymptotic Tight Bound

Θ-Notation For a function $g(n)$,

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

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

Θ-Notation: Asymptotic Tight Bound

Θ-Notation For a function $g(n)$,

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

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

Asymptotic Notations	O	Ω	Θ
Comparison Relations	$\leq$	$\geq$	$=$

Θ-Notation: Asymptotic Tight Bound

Θ-Notation For a function $g(n)$,

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

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

Asymptotic Notations	O	Ω	Θ
Comparison Relations	$\leq$	$\geq$	$=$

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

Recall: O, Ω, Θ-Notation: Asymptotic Bounds

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

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

Ω-**Notation** For a function $g(n)$,

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

Θ-**Notation** For a function $g(n)$,

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

Asymptotic Notations	O	Ω	Θ
Comparison Relations	$\leq$	$\geq$	$=$

Asymptotic Notations	O	Ω	Θ
Comparison Relations	$\leq$	$\geq$	$=$

Trivial Facts on Comparison Relations

- $a \leq b \;\Leftrightarrow\; b \geq a$
- $a = b \;\Leftrightarrow\; a \leq b$ and $a \geq b$
- $a \leq b$ or $a \geq b$

Asymptotic Notations	O	Ω	Θ
Comparison Relations	$\leq$	$\geq$	$=$

Trivial Facts on Comparison Relations

- $a \leq b \;\Leftrightarrow\; b \geq a$
- $a = b \;\Leftrightarrow\; a \leq b$ and $a \geq b$
- $a \leq b$ or $a \geq b$

Correct Analogies

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

Asymptotic Notations	O	Ω	Θ
Comparison Relations	$\leq$	$\geq$	$=$

Trivial Facts on Comparison Relations

- $a \leq b \;\Leftrightarrow\; b \geq a$
- $a = b \;\Leftrightarrow\; a \leq b$ and $a \geq b$
- $a \leq b$ or $a \geq b$

Correct Analogies

- $f(n) = O(g(n)) \;\Leftrightarrow\; g(n) = \Omega(f(n))$
- $f(n) = \Theta(g(n)) \;\Leftrightarrow\; f(n) = O(g(n))$ 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 $f(n) = \Omega(g(n))$

Incorrect Analogy

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

$$f(n) = n^2$$

$$g(n) = \begin{cases} 1 & \text{if } n \text{ is odd} \\ n^3 & \text{if } n \text{ is even} \end{cases}$$

Recall: Informal way to define O-notation

- ignoring lower order terms: $3n^2 - 10n - 5 \to 3n^2$
- ignoring leading constant: $3n^2 \to n^2$

Recall: Informal way to define O-notation

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

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, $3n^2 - 10n - 5 = \Omega(n^2), 3n^2 - 10n - 5 = \Theta(n^2)$
- In the formal definition of $O(\cdot)$, nothing tells us to ignore lower order terms and leading constant.

Recall: Informal way to define O-notation

- ignoring lower order terms: $3n^2 - 10n - 5 \to 3n^2$
- 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

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.

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 Ω and Θ 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, Ω or Θ of g.

f	g	O	Ω	Θ
$n^3 - 100n$	$5n^2 + 3n$			
$3n - 50$	$n^2 - 7n$			
$n^2 - 100n$	$5n^2 + 30n$			
$\log_2 n$	$\log_{10} n$			
$\log^{10} n$	$n^{0.1}$			
2^n	$2^{n/2}$			
$\sqrt{n}$	$n^{\sin n}$			

Exercise

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

f	g	O	Ω	Θ
$n^3 - 100n$	$5n^2 + 3n$	No	Yes	No
$3n - 50$	$n^2 - 7n$			
$n^2 - 100n$	$5n^2 + 30n$			
$\log_2 n$	$\log_{10} n$			
$\log^{10} n$	$n^{0.1}$			
2^n	$2^{n/2}$			
$\sqrt{n}$	$n^{\sin n}$			

Exercise

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

f	g	O	Ω	Θ
$n^3 - 100n$	$5n^2 + 3n$	No	Yes	No
$3n - 50$	$n^2 - 7n$	Yes	No	No
$n^2 - 100n$	$5n^2 + 30n$			
$\log_2 n$	$\log_{10} n$			
$\log^{10} n$	$n^{0.1}$			
2^n	$2^{n/2}$			
$\sqrt{n}$	$n^{\sin n}$			

Exercise

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

f	g	O	Ω	Θ
$n^3 - 100n$	$5n^2 + 3n$	No	Yes	No
$3n - 50$	$n^2 - 7n$	Yes	No	No
$n^2 - 100n$	$5n^2 + 30n$	Yes	Yes	Yes
$\log_2 n$	$\log_{10} n$			
$\log^{10} n$	$n^{0.1}$			
2^n	$2^{n/2}$			
$\sqrt{n}$	$n^{\sin n}$			

Exercise

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

f	g	O	Ω	Θ
$n^3 - 100n$	$5n^2 + 3n$	No	Yes	No
$3n - 50$	$n^2 - 7n$	Yes	No	No
$n^2 - 100n$	$5n^2 + 30n$	Yes	Yes	Yes
$\log_2 n$	$\log_{10} n$	Yes	Yes	Yes
$\log^{10} n$	$n^{0.1}$			
2^n	$2^{n/2}$			
$\sqrt{n}$	$n^{\sin n}$			

We often use $\log n$ for $\log_2 n$. But for $O(\log n)$, the base is not important.

Exercise

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

f	g	O	Ω	Θ
$n^3 - 100n$	$5n^2 + 3n$	No	Yes	No
$3n - 50$	$n^2 - 7n$	Yes	No	No
$n^2 - 100n$	$5n^2 + 30n$	Yes	Yes	Yes
$\log_2 n$	$\log_{10} n$	Yes	Yes	Yes
$\log^{10} n$	$n^{0.1}$	Yes	No	No
2^n	$2^{n/2}$			
$\sqrt{n}$	$n^{\sin n}$			

We often use $\log n$ for $\log_2 n$. But for $O(\log n)$, the base is not important.

Exercise

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

f	g	O	Ω	Θ
$n^3 - 100n$	$5n^2 + 3n$	No	Yes	No
$3n - 50$	$n^2 - 7n$	Yes	No	No
$n^2 - 100n$	$5n^2 + 30n$	Yes	Yes	Yes
$\log_2 n$	$\log_{10} n$	Yes	Yes	Yes
$\log^{10} n$	$n^{0.1}$	Yes	No	No
2^n	$2^{n/2}$	No	Yes	No
$\sqrt{n}$	$n^{\sin n}$			

We often use $\log n$ for $\log_2 n$. But for $O(\log n)$, the base is not important.

Exercise

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

f	g	O	Ω	Θ
$n^3 - 100n$	$5n^2 + 3n$	No	Yes	No
$3n - 50$	$n^2 - 7n$	Yes	No	No
$n^2 - 100n$	$5n^2 + 30n$	Yes	Yes	Yes
$\log_2 n$	$\log_{10} n$	Yes	Yes	Yes
$\log^{10} n$	$n^{0.1}$	Yes	No	No
2^n	$2^{n/2}$	No	Yes	No
$\sqrt{n}$	$n^{\sin n}$	No	No	No

We often use $\log n$ for $\log_2 n$. But for $O(\log n)$, the base is not important.

Asymptotic Notations	O	Ω	Θ	o	ω
Comparison Relations	$\leq$	$\geq$	$=$	$<$	$>$

Asymptotic Notations	O	Ω	Θ	o	ω
Comparison Relations	$\leq$	$\geq$	$=$	$<$	$>$

Questions?

Outline

$O(n)$ (Linear) Running Time

Computing the sum of n numbers

sum(A, n)

```
S ← 0
for i ← 1 to n
    S ← S + A[i]
return S
```

$O(n)$ (Linear) Running Time

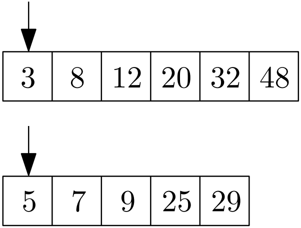
- Merge two sorted arrays

3	8	12	20	32	48

5	7	9	25	29

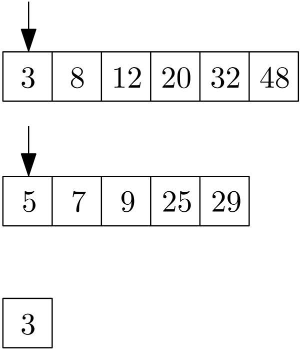
$O(n)$ (Linear) Running Time

- Merge two sorted arrays



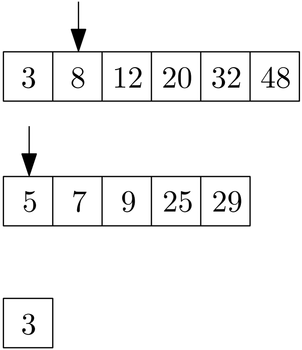
$O(n)$ (Linear) Running Time

- Merge two sorted arrays



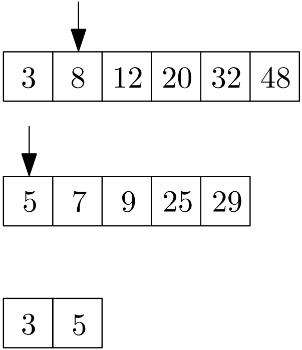
$O(n)$ (Linear) Running Time

- Merge two sorted arrays



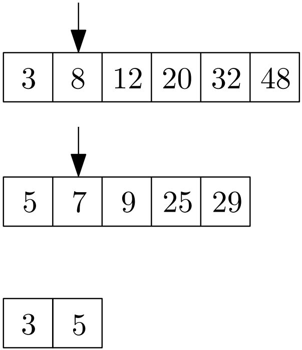
$O(n)$ (Linear) Running Time

- Merge two sorted arrays



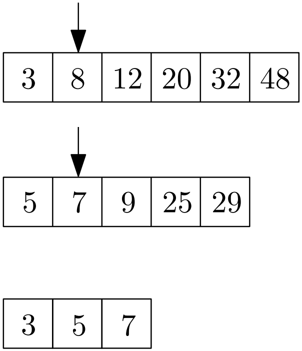
$O(n)$ (Linear) Running Time

- Merge two sorted arrays



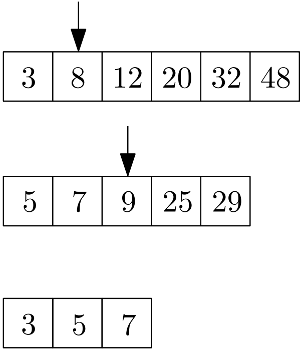
$O(n)$ (Linear) Running Time

- Merge two sorted arrays



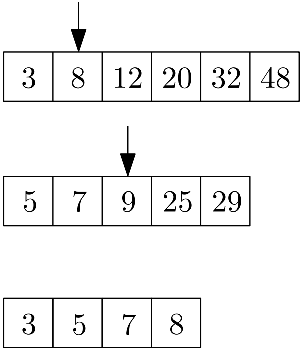
$O(n)$ (Linear) Running Time

- Merge two sorted arrays



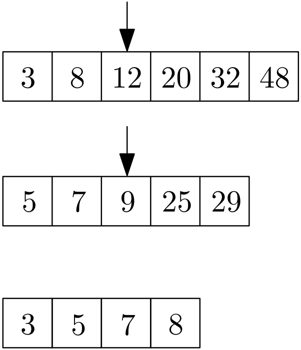
$O(n)$ (Linear) Running Time

- Merge two sorted arrays



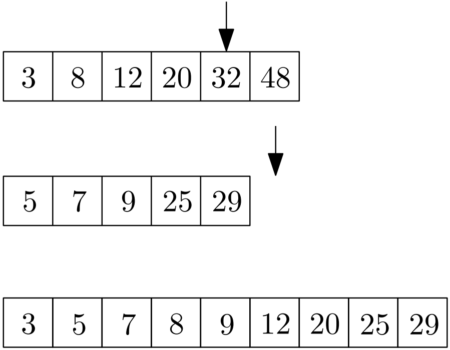
$O(n)$ (Linear) Running Time

- Merge two sorted arrays



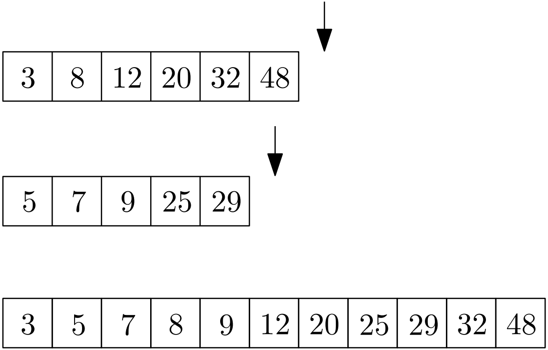
$O(n)$ (Linear) Running Time

- Merge two sorted arrays



$O(n)$ (Linear) Running Time

- Merge two sorted arrays



$O(n)$ (Linear) Running Time

merge(B, C, n_1, n_2) \\ B and C are sorted, with length n_1 and n_2

$A \leftarrow []$; $i \leftarrow 1$; $j \leftarrow 1$
while $i \leq n_1$ and $j \leq n_2$ **do**
  if $B[i] \leq C[j]$ **then**
    append $B[i]$ to A; $i \leftarrow i + 1$
  else
    append $C[j]$ to A; $j \leftarrow j + 1$
if $i \leq n_1$ then append $B[i..n_1]$ to A
if $j \leq n_2$ then append $C[j..n_2]$ to A
return A

$O(n)$ (Linear) Running Time

merge(B, C, n_1, n_2) \\ B and C are sorted, with length n_1 and n_2

```
A ← []; i ← 1; j ← 1
while i ≤ n1 and j ≤ n2 do
    if B[i] ≤ C[j] then
        append B[i] to A; i ← i + 1
    else
        append C[j] to A; j ← j + 1
if i ≤ n1 then append B[i..n1] to A
if j ≤ n2 then append C[j..n2] to A
return A
```

Running time $= O(n)$ where $n = n_1 + n_2$.

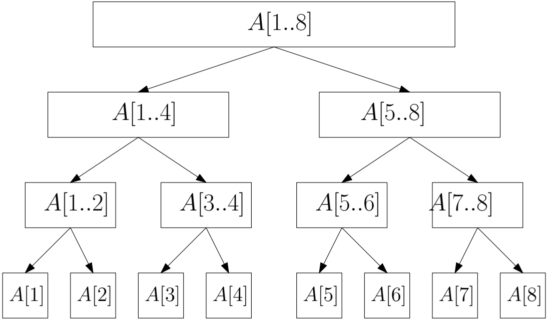
$O(n \log n)$ Running Time

merge-sort(A, n)

if $n = 1$ **then**
    return A
$B \leftarrow$ merge-sort$\left(A\left[1..\lfloor n/2 \rfloor\right], \lfloor n/2 \rfloor\right)$
$C \leftarrow$ merge-sort$\left(A\left[\lfloor n/2 \rfloor + 1..n\right], n - \lfloor n/2 \rfloor\right)$
return merge$(B, C, \lfloor n/2 \rfloor, n - \lfloor n/2 \rfloor)$

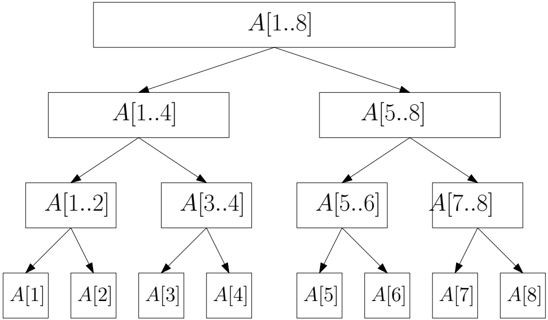
$O(n \log n)$ Running Time

- Merge-Sort



$O(n \log n)$ Running Time

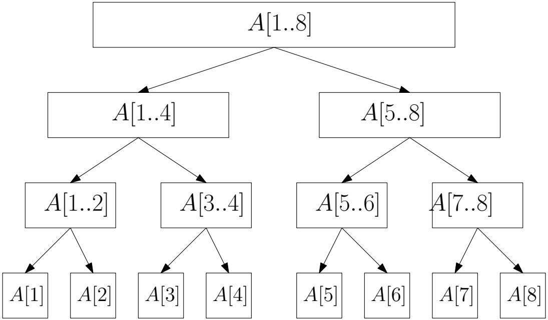
- Merge-Sort



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

$O(n \log n)$ Running Time

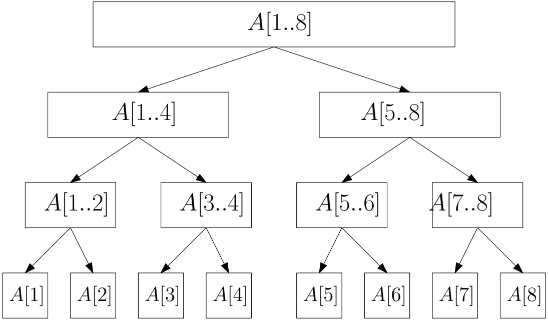
- Merge-Sort



- Each level takes running time $O(n)$
- There are $O(\log n)$ levels

$O(n \log n)$ Running Time

- Merge-Sort



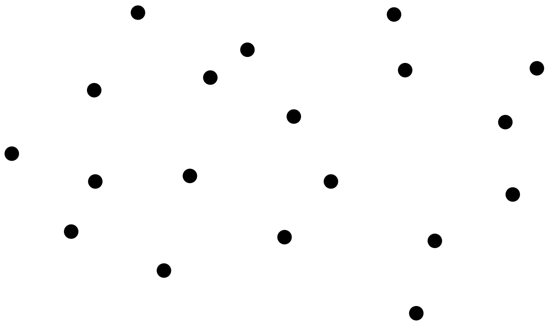
- Each level takes running time $O(n)$
- There are $O(\log n)$ levels
- Running time $= O(n \log n)$

$O(n^2)$ (Quardatic) 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

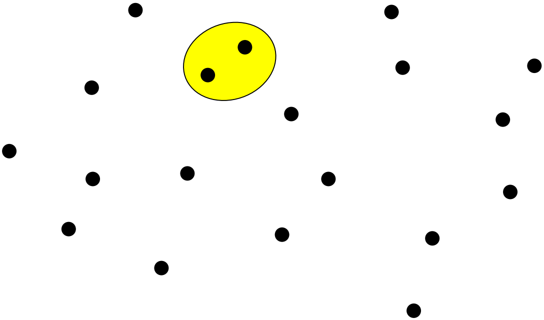


$O(n^2)$ (Quardatic) Running Time

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



$O(n^2)$ (Quardatic) Running Time

Closest Pair

Input: n points in plane: $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$

Output: the pair of points that are closest

closest-pair(x, y, n)

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

$O(n^2)$ (Quardatic) Running Time

Closest Pair

Input: n points in plane: $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$

Output: the pair of points that are closest

closest-pair(x, y, n)

```
1: bestd ← ∞
2: for i ← 1 to n − 1 do
3:     for j ← i + 1 to n do
4:         d ← √((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 can be solved in $O(n \log n)$ time!

$O(n^3)$ (Cubic) Running Time

Multiply two matrices of size $n \times n$

matrix-multiplication(A, B, n)

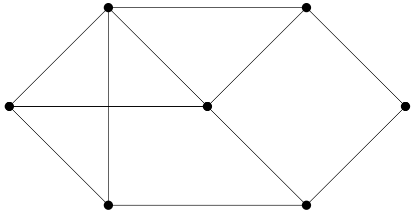
```
C ← matrix of size n × n, with all entries being 0
for i ← 1 to n do
    for j ← 1 to n do
        for k ← 1 to n do
            C[i, k] ← C[i, k] + A[i, j] × B[j, k]
return C
```

Beyond Polynomial Time: 2^n

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 $(u, v) \notin E$.

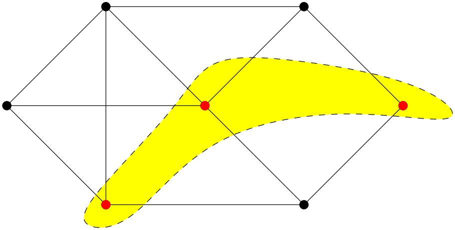
Beyond Polynomial Time: 2^n

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 $(u, v) \notin E$.



Beyond Polynomial Time: 2^n

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 $(u, v) \notin E$.



Beyond Polynomial Time: 2^n

Maximum Independent Set Problem

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

Output: the maximum independent set of G

Beyond Polynomial Time: 2^n

Maximum Independent Set Problem

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

Output: the maximum independent set of G

max-independent-set($G = (V, E)$)

$R \leftarrow \emptyset$
for every set $S \subseteq V$ **do**
    $b \leftarrow$ true
    for every $u, v \in S$ **do**
        if $(u, v) \in E$ then $b \leftarrow$ false
    if b and $|S| > |R|$ then $R \leftarrow S$
return R

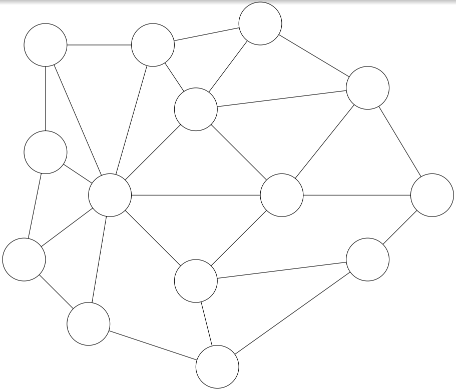
Running time $= O(2^n n^2)$.

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

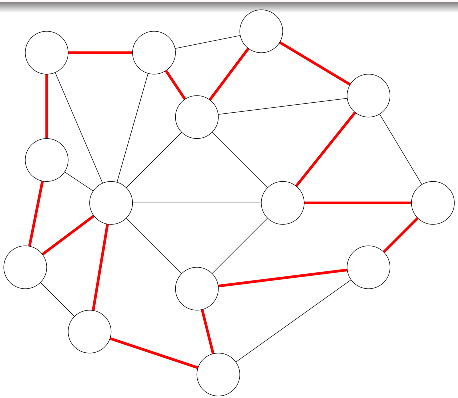


Beyond Polynomial Time: $n!$

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

for every permutation $(p_1, p_2, \cdots, p_n)$ of V **do**
    $b \leftarrow$ true
    for $i \leftarrow 1$ to $n - 1$ **do**
        if $(p_i, p_{i+1}) \notin E$ then $b \leftarrow$ false
    if $(p_n, p_1) \notin E$ then $b \leftarrow$ false
    if b then return $(p_1, p_2, \cdots, p_n)$
return "No Hamiltonian Cycle"

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

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

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

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

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

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

3	8	10	25	29	37	38	42	46	52	59	61	63	75	79

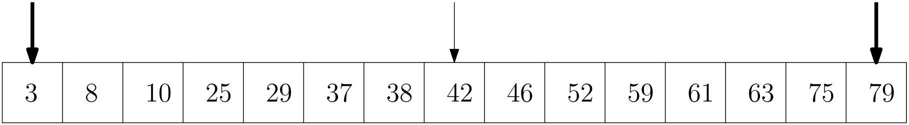
$O(\log n)$ (Logarithmic) Running Time

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

3	8	10	25	29	37	38	42	46	52	59	61	63	75	79

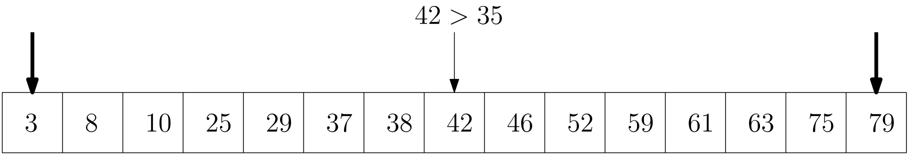
$O(\log n)$ (Logarithmic) Running Time

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



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

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

3	8	10	25	29	37	38	42	46	52	59	61	63	75	79

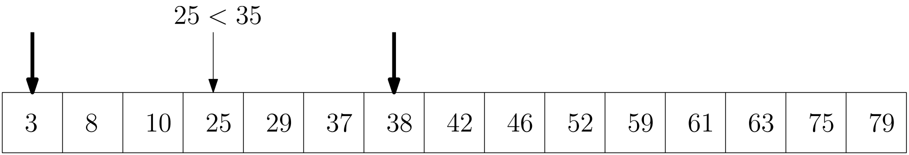
$O(\log n)$ (Logarithmic) Running Time

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

3	8	10	25	29	37	38	42	46	52	59	61	63	75	79

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

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



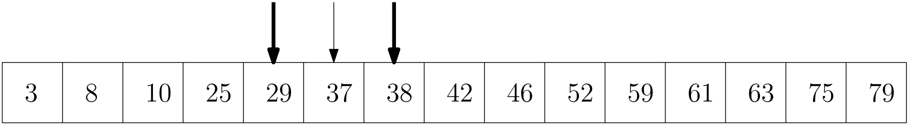
$O(\log n)$ (Logarithmic) Running Time

- 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

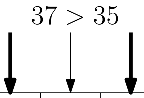
$O(\log n)$ (Logarithmic) Running Time

- 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:



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

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



3	8	10	25	29	37	38	42	46	52	59	61	63	75	79

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

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

3	8	10	25	29	37	38	42	46	52	59	61	63	75	79

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

Binary search

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

binary-search(A, n, t)

```
i ← 1, j ← n
while i ≤ j do
    k ← ⌊(i + j)/2⌋
    if A[k] = t return true
    if t < A[k] then j ← k − 1 else i ← k + 1
return false
```

$O(\log n)$ (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)

```
i ← 1, j ← n
while i ≤ j do
    k ← ⌊(i + j)/2⌋
    if A[k] = t return true
    if t < A[k] then j ← k − 1 else i ← k + 1
return false
```

Running time $= O(\log n)$

Comparing the Orders

- Sort the functions from smallest to largest asymptotically
  $\log n, \;\; n \log n, \;\; n, \;\; n!, \;\; n^2, \;\; 2^n, \;\; e^n, \;\; n^n$
- $\log n = O(n)$

Comparing the Orders

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

Comparing the Orders

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

Comparing the Orders

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

Comparing the Orders

- Sort the functions from smallest to largest asymptotically
  $\log n, \;\; n \log n, \;\; n, \;\; n!, \;\; n^2, \;\; 2^n, \;\; e^n, \;\; n^n$
- $\log n = O(n)$
- $n = O(n \log n)$
- $n \log n = O(n^2)$
- $n^2 = O(2^n)$
- $2^n = O(n!)$

Comparing the Orders

- Sort the functions from smallest to largest asymptotically
  $\log n, \quad n \log n, \quad n, \quad n!, \quad n^2, \quad 2^n, \quad e^n, \quad n^n$
- $\log n = O(n)$
- $n = O(n \log n)$
- $n \log n = O(n^2)$
- $n^2 = O(2^n)$
- $2^n = O(e^n)$
- $e^n = O(n!)$

Comparing the Orders

- Sort the functions from smallest to largest asymptotically
  $\log n, \;\; n \log n, \;\; n, \;\; n!, \;\; n^2, \;\; 2^n, \;\; e^n, \;\; n^n$
- $\log n = O(n)$
- $n = O(n \log n)$
- $n \log n = O(n^2)$
- $n^2 = O(2^n)$
- $2^n = O(e^n)$
- $e^n = O(n!)$
- $n! = O(n^n)$

Terminologies

When we talk about upper 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: $o(n^2)$