CSE 431/531: Algorithm Analysis and Design (Spring 2018)

Introduction and Syllabus

Lecturer: Shi Li

Department of Computer Science and Engineering
University at Buffalo

Outline

- Course Webpage (contains schedule, policies, homeworks and slides):
  http://www.cse.buffalo.edu/~shil/courses/CSE531/
- Please sign up the course on Piazza from course webpage polls, asking/answering questions.

- Time and locatiion:
  - MoWeFr, 9:00-9:50am
  - Talbert 107
- Lecturer:
  - Shi Li, shil@buffalo.edu
  - Office hours: TBD
- TAs
  - TBD

You should know:

You should know:

- Mathematical Tools
  - Mathematical inductions
  - Probabilities and random variables

You should know:

- Mathematical Tools
  - Mathematical inductions
  - Probabilities and random variables
- Data Structures
  - Stacks, queues, linked lists

You should know:

- Mathematical Tools
  - Mathematical inductions
  - Probabilities and random variables
- Data Structures
  - Stacks, queues, linked lists
- Some Programming Experience
  - E.g., 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

Tentative Schedule

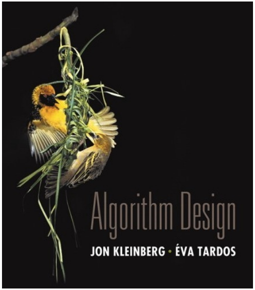
- Introduction, 3 lectures
- Basic Graph Algorithms, 3 lectures
- Greedy Algorihtms, 6 lectures (include recitation)
- Divide and Conquer, 6 lectures (include recitation)
- In-Class Exam #1, Mar 12, 2018
- Dynamic Programming, 6 lectures (include recitation)
- Linear Programming, 6 lectures (include recitation)
- In-Class Exam #2, Apr 18, 2018
- NP-Completeness 6 lectures (include recitation)
- Final Review, 1 lecture

- Exercise problems will be posted before each recitation class

Textbook

Textbook (Highly Recommended):

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



Other Reference Books

- Introduction to Algorithms, Third Edition, *Thomas Cormen, Charles Leiserson, Rondald Rivest, Clifford Stein*

Reading Before Classes

- Highly recommended: read the correspondent sections from the textbook (or reference book) before classes
- Slides will be posted online before class

Grading

- 40% for homeworks
  - 6 homeworks, 5 of which contain programming problems

Grading

- 40% for homeworks
  - 6 homeworks, 5 of which contain programming problems
- 60% for two in-class exams + final exam

$$
\begin{aligned}
&\max\{E1 \times 5\% + F \times 25\%, E1 \times 15\% + F \times 15\%\} \\
&+ \max\{E2 \times 5\% + F \times 25\%, E2 \times 15\% + F \times 15\%\} \\
&E1, E2, F \in [0, 100]
\end{aligned}
$$

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 discussing
  - 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 courses
- Copy solutions from other students

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 courses
- Copy solutions from other students

If cheating is found, you will get an "F" for the course. The case will be reported to the department.

For Programming Problems

- Need to implement the algorithms by your self
- Can not copy codes from others or the Internet

For Programming Problems

- Need to implement the algorithms by your self
- Can not copy codes from others or the Internet

If cheating is found, you will get an "F" for the course. The case will be reported to the department.

Late policy

- You have one late credit
- turn in a homework late for three days using the late credit
- no other late submissions will be accepted

Exams

- Closed-book
- Can bring one A4 handwritten sheet

Exams

- Closed-book
- Can bring one A4 handwritten sheet

If you are caught cheating in exams, you will get an “F” for the course. The case will be reported to the department.

Exams

- Closed-book
- Can bring one A4 handwritten sheet

If you are caught cheating in exams, you will get an "F" for the course. The case will be reported to the department.

Questions?

Outline

Outline

What is an Algorithm?

- 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
- $\gcd(270, 210) = \gcd(210, 270 \bmod 210) = \gcd(210, 60)$

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
- $\gcd(270, 210) = \gcd(210, 270 \bmod 210) = \gcd(210, 60)$
- $(270, 210) \to (210, 60) \to (60, 30) \to (30, 0)$

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 $a'_1 \le a'_2 \le \cdots \le a'_n$

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 $a'_1 \le a'_2 \le \cdots \le a'_n$

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 $a'_1 \le a'_2 \le \cdots \le a'_n$

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 $G = (V, E)$, $s, t \in V$

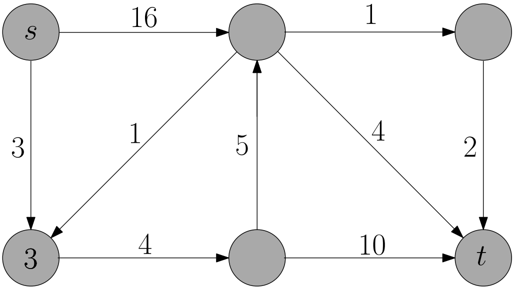
Output: a shortest path from s to t in G

Examples

Shortest Path

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

Output: a shortest path from s to t in G

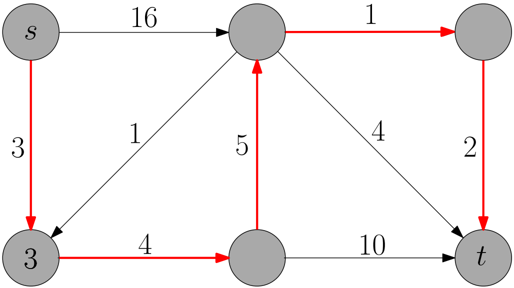


Examples

Shortest Path

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

Output: a shortest path from s to t in G

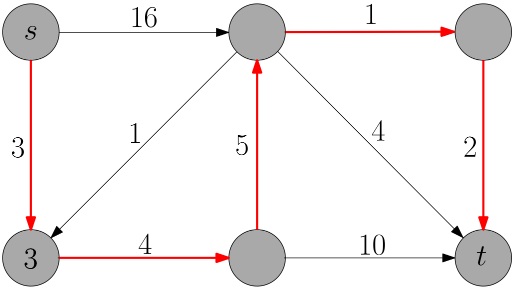


Examples

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

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

Pseudo-Code:

Euclidean(a, b)

1. while $b > 0$
2. $\quad (a, b) \leftarrow (b, a \bmod b)$
3. return a

C++ program:

```
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)

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

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, make the first j numbers 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 ← 2 to n
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← 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. $\quad key \leftarrow A[j]$
3. $\quad i \leftarrow j - 1$
4. $\quad$ while $i > 0$ and $A[i] > key$
5. $\qquad A[i+1] \leftarrow A[i]$
6. $\qquad i \leftarrow i - 1$
7. $\quad A[i+1] \leftarrow key$

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

$$\begin{array}{cccccc} 12 & 21 & 35 & 53 & 59 & 15 \\ & & & & \uparrow & \\ & & & & i & \end{array}$$

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
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$

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

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

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

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

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

Example:

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

insertion-sort(A, n)

```
for j ← 2 to n
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← key
```

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

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

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
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$

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

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

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
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$

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

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

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
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow key$

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

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

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

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

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

Example:

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

insertion-sort(A, n)

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

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

$$\begin{array}{cccccc} 12 & \color{red}{15} & 21 & 35 & 53 & 59 \\ \uparrow & & & & & \\ i & & & & & \end{array}$$

Outline

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 $A[1..j]$.

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

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

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

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

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

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

Analyze Running Time of Insertion Sort

- Q: Size of input?

Analyze Running Time of Insertion Sort

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

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?

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

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?

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?
- 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)

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

Asymptotic Analysis of Insertion Sort

insertion-sort(A, n)

```
for j ← 2 to n
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key
        A[i + 1] ← A[i]
        i ← i − 1
    A[i + 1] ← key
```

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

Asymptotic Analysis of Insertion Sort

insertion-sort(A, n)

```
for j ← 2 to n
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key
        A[i + 1] ← A[i]
        i ← i − 1
    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)

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

- 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)

Computation Model

- Random-Access Machine (RAM) model: read $A[j]$ takes $O(1)$ time.
- Basic operations take $O(1)$ time: addition, subtraction, multiplication, etc.
- Each integer (word) has $c \log n$ bits, $c \geq 1$ large enough

Computation Model

- Random-Access Machine (RAM) model: read $A[j]$ takes $O(1)$ time.
- 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

- Random-Access Machine (RAM) model: read $A[j]$ takes $O(1)$ time.
- 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

- Random-Access Machine (RAM) model: read $A[j]$ takes $O(1)$ time.
- 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?

Computation Model

- Random-Access Machine (RAM) model: read $A[j]$ takes $O(1)$ time.
- 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

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 $f(n) > 0$

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 $g(n)$,

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

O-Notation: Asymptotic Upper Bound

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

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

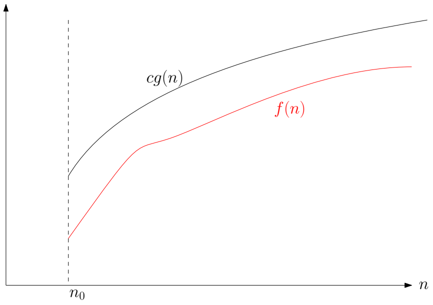
- In other words, $f(n) \in O(g(n))$ if $f(n) \leq cg(n)$ for some c and large enough n.

O-Notation: Asymptotic Upper Bound

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

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

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



O-Notation: Asymptotic Upper Bound

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

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

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

O-Notation: Asymptotic Upper Bound

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

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

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

Proof.

Let $c = 4$ and $n_0 = 10$, for every $n > n_0 = 50$, we have,

$$3n^2 + 2n - c(n^2 - 10n) = 3n^2 + 2n - 4(n^2 - 10n)$$
$$= -n^2 + 40n \le 0.$$
$$3n^2 + 2n \le c(n^2 - 10n)$$

□

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

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

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

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.

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

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 $g(n)$,

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

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

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

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.

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

Asymptotic Notations	O	Ω	Θ
Comparison Relations	$\leq$		

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 statements are wrong:

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 statements are wrong:

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

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 $g(n)$,

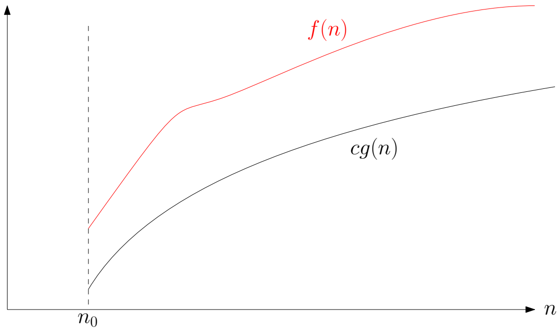
$$\Omega(g(n)) = \{\text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \geq cg(n), \forall n \geq n_0\}.$$

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

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 $g(n)$,

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

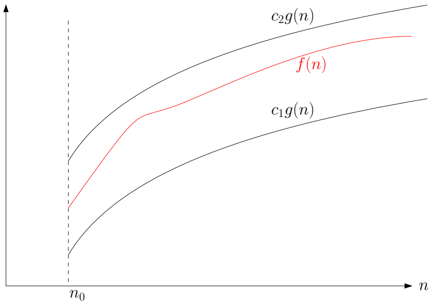
- $f(n) = \Theta(g(n))$, then for large enough n, we have "$f(n) \approx g(n)$".

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 $g(n)$,

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

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

Θ-Notation: Asymptotic Tight Bound

Θ-Notation For a function $g(n)$,

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

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

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

Θ-Notation: Asymptotic Tight Bound

Θ-Notation For a function $g(n)$,

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

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

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 $f(n) = \Omega(g(n))$.

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
$n^3 - 100n$	$5n^2 + 3n$			
$3n - 50$	$n^2 - 7n$			
$n^2 - 100n$	$5n^2 + 30n$			
$\lg^{10} n$	$n^{0.1}$			
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
$3n - 50$	$n^2 - 7n$			
$n^2 - 100n$	$5n^2 + 30n$			
$\lg^{10} n$	$n^{0.1}$			
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
$n^2 - 100n$	$5n^2 + 30n$			
$\lg^{10} n$	$n^{0.1}$			
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
$\lg^{10} n$	$n^{0.1}$			
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
$\lg^{10} n$	$n^{0.1}$	Yes	No	No
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
$\lg^{10} n$	$n^{0.1}$	Yes	No	No
2^n	$2^{n/2}$	No	Yes	No
$\sqrt{n}$	$n^{\sin n}$			

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
$\lg^{10} n$	$n^{0.1}$	Yes	No	No
2^n	$2^{n/2}$	No	Yes	No
$\sqrt{n}$	$n^{\sin n}$	No	No	No

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

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

Trivial Facts on Comparison Relations

- $f \leq g \;\Leftrightarrow\; g \geq f$
- $f = g \;\Leftrightarrow\; f \leq g$ and $f \geq g$
- $f \leq g$ or $f \geq g$

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

Trivial Facts on Comparison Relations

- $f \leq g \;\Leftrightarrow\; g \geq f$
- $f = g \;\Leftrightarrow\; f \leq g$ and $f \geq g$
- $f \leq g$ or $f \geq g$

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

- $f \leq g \;\Leftrightarrow\; g \geq f$
- $f = g \;\Leftrightarrow\; f \leq g$ and $f \geq g$
- $f \leq g$ or $f \geq g$

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

Incorrect Analogy

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

Incorrect Analogy

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

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

$$g(n) = \begin{cases} 1 & \text{if } n \text{ is odd} \\ 2^n & \text{if } n \text{ is even} \end{cases}$$

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, $3n^2 - 10n - 5 = \Omega(n^2), 3n^2 - 10n - 5 = \Theta(n^2)$
- theoretically, nothing tells us to ignore lower order terms and leading constant.

Recall: informal way to define O-notation

- ignoring lower order terms: $3n^2 - 10n - 5 \to 3n^2$
- 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.
- $3n^2 - 10n - 5 = O(5n^2 - 6n + 5)$ is correct, although 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, $3n^2 - 10n - 5 = \Omega(n^2), 3n^2 - 10n - 5 = \Theta(n^2)$

- theoretically, nothing tells us to ignore lower order terms and leading constant.
- $3n^2 - 10n - 5 = O(5n^2 - 6n + 5)$ is correct, although weird.
- $3n^2 - 10n - 5 = O(n^2)$ is simpler.

o and ω-Notations

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

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

ω-Notation For a function $g(n)$,

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

Example:

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

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)

1. $S \leftarrow 0$
2. for $i \leftarrow 1$ to n
3. $\quad S \leftarrow S + A[i]$
4. return S

$O(n)$ (Linear) Running Time

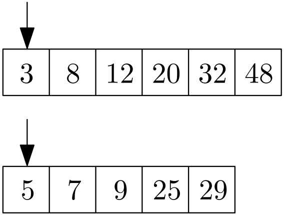
- Merge two sorted arrays

3	8	12	20	32	48

5	7	9	25	29

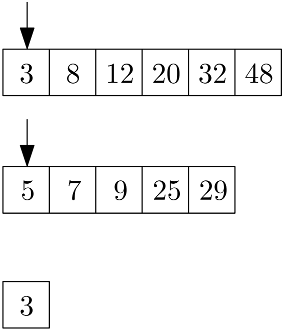
$O(n)$ (Linear) Running Time

- Merge two sorted arrays



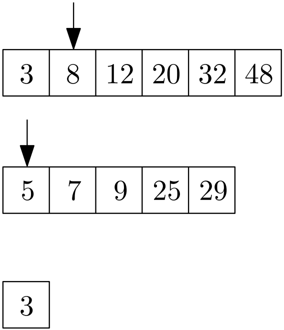
$O(n)$ (Linear) Running Time

- Merge two sorted arrays



$O(n)$ (Linear) Running Time

- Merge two sorted arrays



$O(n)$ (Linear) Running Time

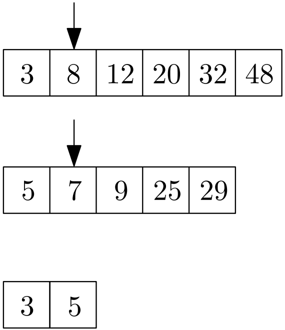
- Merge two sorted arrays

3	8	12	20	32	48

5	7	9	25	29

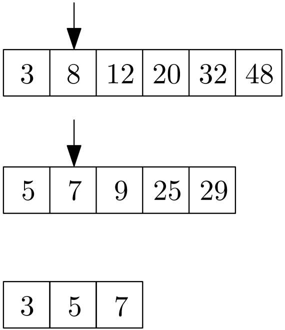
3	5

- Merge two sorted arrays



$O(n)$ (Linear) Running Time

- Merge two sorted arrays



$O(n)$ (Linear) Running Time

- Merge two sorted arrays

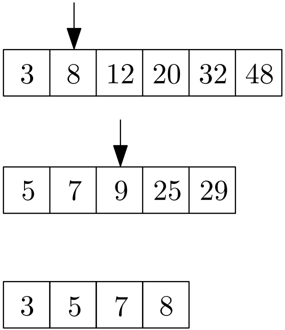
3	8	12	20	32	48

5	7	9	25	29

3	5	7

$O(n)$ (Linear) Running Time

- Merge two sorted arrays



$O(n)$ (Linear) Running Time

- Merge two sorted arrays

3	8	12	20	32	48

5	7	9	25	29

3	5	7	8

$O(n)$ (Linear) Running Time

- Merge two sorted arrays

3	8	12	20	32	48

5	7	9	25	29

3	5	7	8	9	12	20	25	29

$O(n)$ (Linear) Running Time

- Merge two sorted arrays

3	8	12	20	32	48

5	7	9	25	29

3	5	7	8	9	12	20	25	29	32	48

$O(n)$ (Linear) Running Time

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

$A \leftarrow []$; $i \leftarrow 1$; $j \leftarrow 1$
while $i \le n_1$ and $j \le n_2$
    if $(B[i] \le C[j])$ then
        append $B[i]$ to A; $i \leftarrow i + 1$
    else
        append $C[j]$ to A; $j \leftarrow j + 1$
if $i \le n_1$ then append $B[i..n_1]$ to A
if $j \le n_2$ 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
    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 $n = n_1 + n_2$.

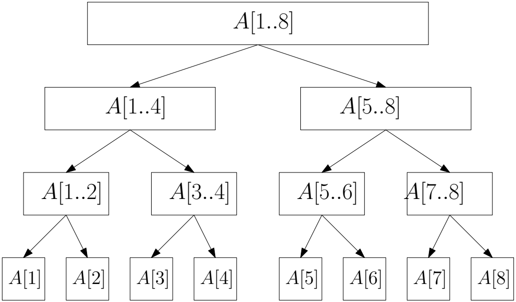
$O(n \lg n)$ Running Time

merge-sort(A, n)

```
if n = 1 then
    return A
else
    B ← merge-sort(A[1..⌊n/2⌋], ⌊n/2⌋)
    C ← merge-sort(A[⌊n/2⌋ + 1..n], n − ⌊n/2⌋)
return merge(B, C, ⌊n/2⌋, n − ⌊n/2⌋)
```

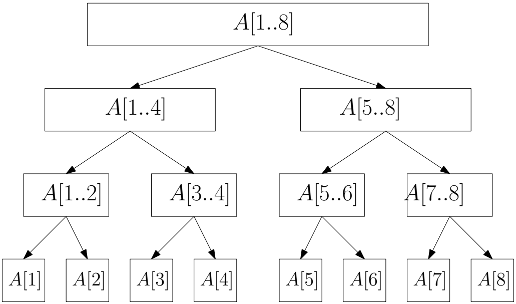
$O(n \lg n)$ Running Time

- Merge-Sort



$O(n \lg n)$ Running Time

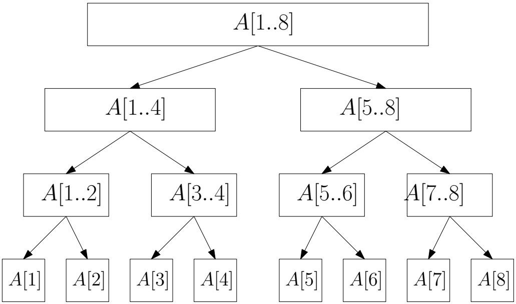
- Merge-Sort



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

$O(n \lg n)$ Running Time

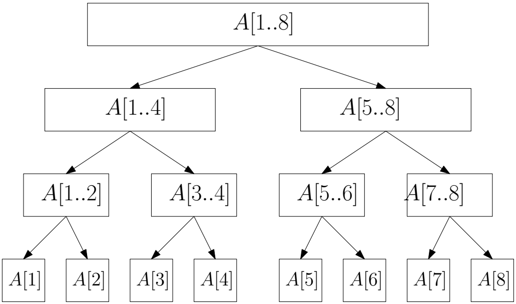
- Merge-Sort



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

$O(n \lg n)$ Running Time

- Merge-Sort



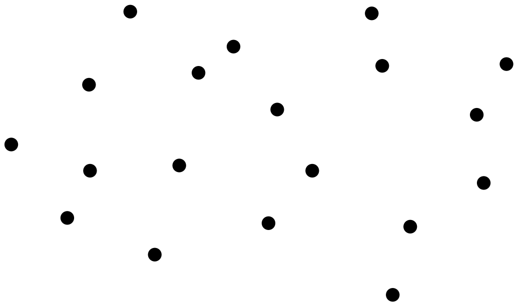
- Each level takes running time $O(n)$
- There are $O(\lg n)$ levels
- Running time $= O(n \lg n)$

$O(n^2)$ (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

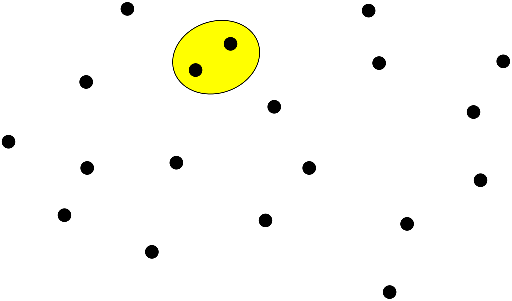


$O(n^2)$ (Quardatic) Running Time

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)

1. $bestd \leftarrow \infty$
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 $d < bestd$ then
6. $besti \leftarrow i, bestj \leftarrow j, bestd \leftarrow d$
7. 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)

```
bestd ← ∞
for i ← 1 to n − 1
    for j ← i + 1 to n
        d ← sqrt((x[i] − x[j])^2 + (y[i] − y[j])^2)
        if d < bestd then
            besti ← i, bestj ← j, bestd ← d
return (besti, bestj)
```

Closest pair can be solved in $O(n \lg n)$ time!

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

Multiply two matrices of size $n \times n$

matrix-multiplication(A, B, n)

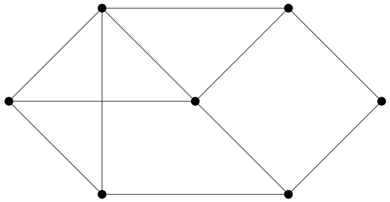
$C \leftarrow$ matrix of size $n \times n$, with all entries being 0
for $i \leftarrow 1$ to n
    for $j \leftarrow 1$ to n
        for $k \leftarrow 1$ to n
            $C[i,k] \leftarrow C[i,k] + A[i,j] \times B[j,k]$
return C

$O(n^k)$ Running Time for Integer $k \geq 4$

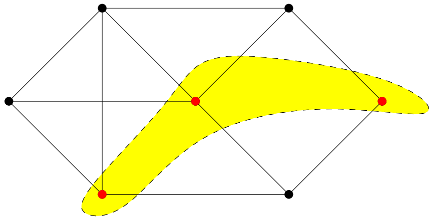
Def. An independent set of a graph $G = (V, E)$ is a subset $S \subseteq V$ of vertices such that for every $u, v \in S$, we have $(u, v) \notin E$.

$O(n^k)$ 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 $(u, v) \notin E$.

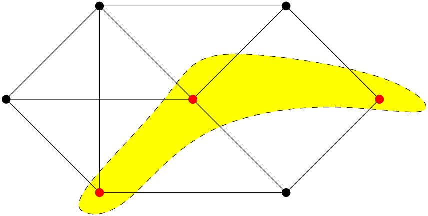


Def. An independent set of a graph $G = (V, E)$ is a subset $S \subseteq V$ of vertices such that for every $u, v \in S$, we have $(u, v) \notin E$.



$O(n^k)$ 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

$O(n^k)$ Running Time for Integer $k \geq 4$

Independent Set of Size k

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

Output: whether there is an independent set of size k

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

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

Running time $= O(\frac{n^k}{k!} \times k^2) = O(n^k)$ (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

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

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

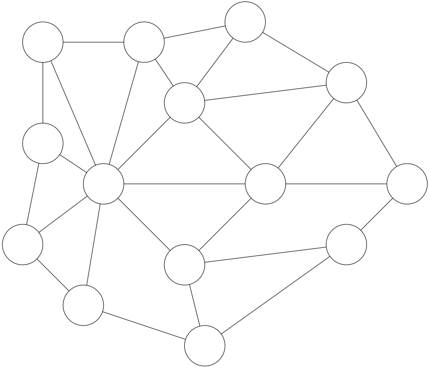
Running time $= O(2^n n^2)$.

Beyond Polynomial Time: $O(n!)$

Hamiltonian Cycle Problem

Input: a graph with n vertices

Output: a cycle that visits each node exactly once,
or say no such cycle exists

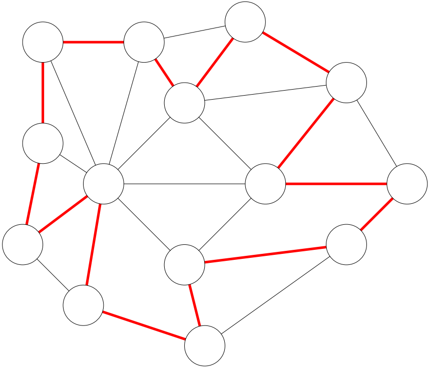


Beyond Polynomial Time: $O(n!)$

Hamiltonian Cycle Problem

Input: a graph with n vertices

Output: a cycle that visits each node exactly once,
or say no such cycle exists



Beyond Polynomial Time: $n!$

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

1. for every permutation $(p_1, p_2, \cdots, p_n)$ of V
2. $\quad b \leftarrow$ true
3. $\quad$ for $i \leftarrow 1$ to $n - 1$
4. $\quad\quad$ if $(p_i, p_{i+1}) \notin E$ then $b \leftarrow$ false
5. $\quad$ if $(p_n, p_1) \notin E$ then $b \leftarrow$ false
6. $\quad$ 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

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

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

$O(\lg n)$ (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

$O(\lg n)$ (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:

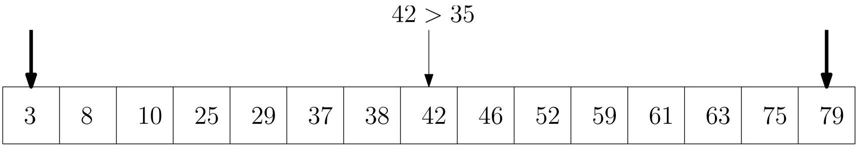
3	8	10	25	29	37	38	42	46	52	59	61	63	75	79

$O(\lg n)$ (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:



$O(\lg n)$ (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

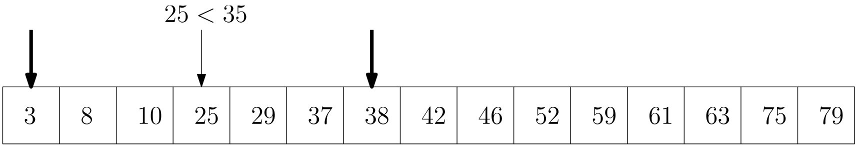
$O(\lg n)$ (Logarithmic) Running Time

- 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

$O(\lg n)$ (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:

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

$O(\lg n)$ (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

$O(\lg n)$ (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:

$37 > 35$

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

$O(\lg n)$ (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

$O(\lg n)$ (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)

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

$O(\lg n)$ (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. $\quad k \leftarrow \lfloor (i+j)/2 \rfloor$
4. $\quad$ if $A[k] = t$ return true
5. $\quad$ 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 $f = \Theta(g)$!
- $\lg n < n^{\sqrt{n}}$

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 $f = \Theta(g)$!
- $\lg n < n^{\sqrt{n}}$
- $\lg n < n < n^{\sqrt{n}}$

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: $o(n^2)$

Terminologies

When we talk about upper bounds:

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

When we talk about lower bounds:

- Super-linear time: $\omega(n)$
- Super-quadratic time: $\omega(n^2)$
- Super-polynomial time: $\bigcap_{k>0} \omega(n^k)$

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, complier 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.