

Announcement

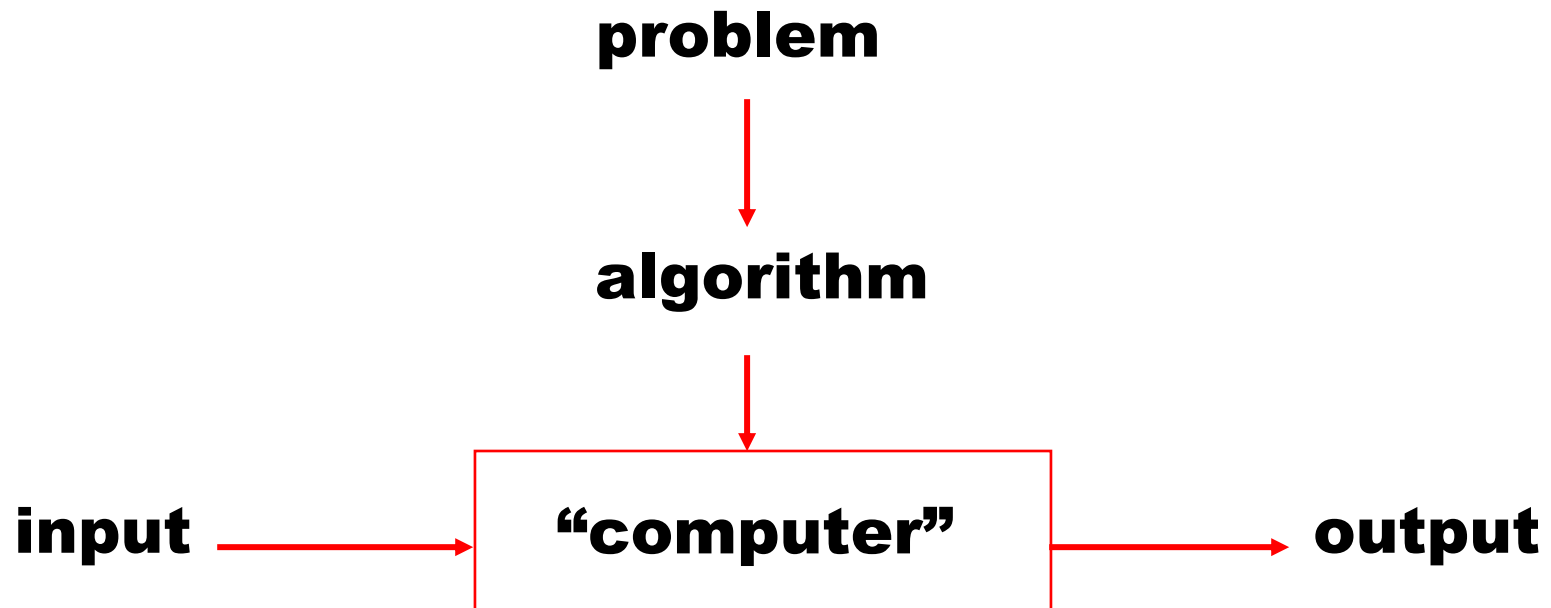
Homework 1 was posted in Blackboard and class website

Due on **Thursday, Jan. 20, before class starts**

You need to submit your homework through Blackboard.

Last Class: Definition of Algorithm

An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input, in a finite amount of time.



Fundamental Data Structure

Data Structure: a particular scheme of organizing related data items

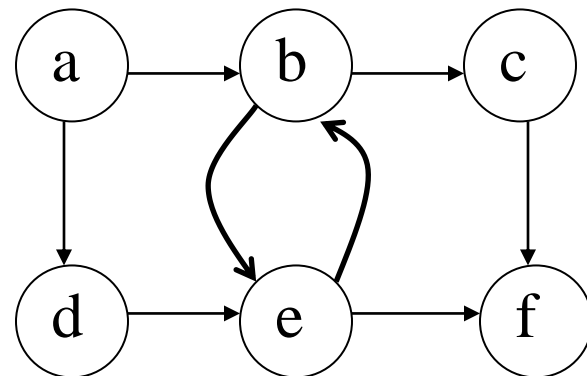
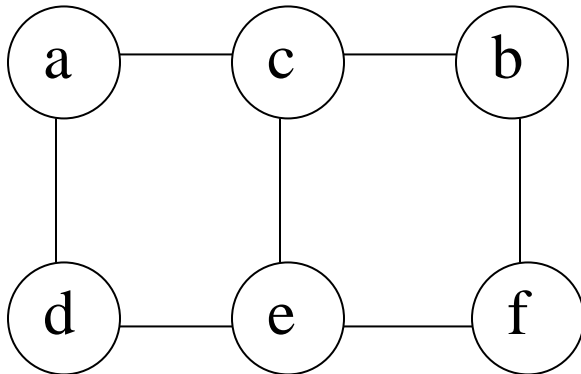
Important data structures:

- Array
- Linked list (queue, stack, heap, ...)
- Graph
- Tree
- Set

Graph

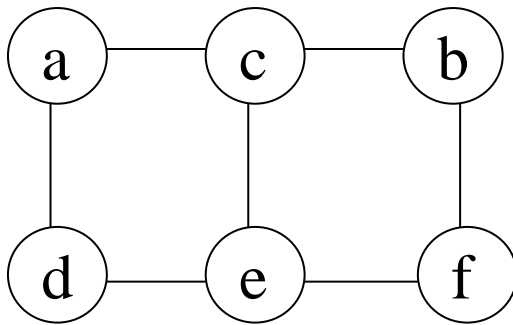
- **Undirected and directed (digraph): $G=\langle V,E\rangle$**
 - vertices (V) and edges (E)
 - number of vertices ($|V|$) and number of edges ($|E|$)
- Undirected graph
 - Endpoints: e.g., two endpoints of an edge (a,b)
- Directed graph/Digraph
 - Tail and head: e.g., tail (a) and head (b) of an edge (a,b)

Write down the V and E for the following two graphs



Graph Representation

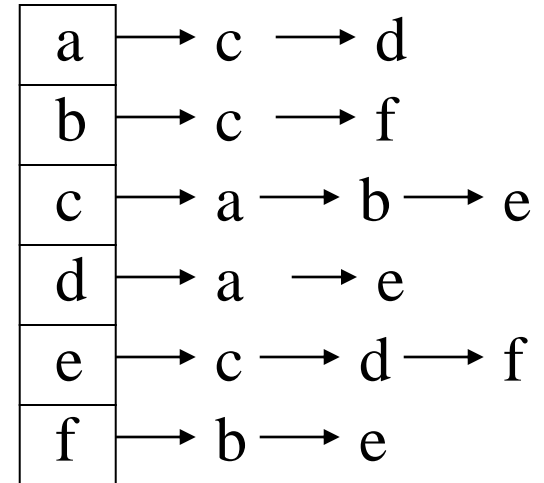
Adjacency matrix and adjacency linked list



Adjacency matrix

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |

Adjacency linked list



When should we use adjacency matrix and when adjacent linked list?

Graph sparseness

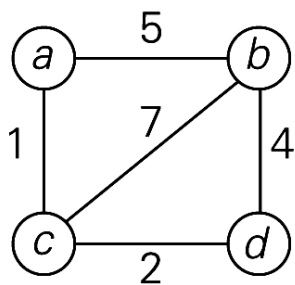
Weighted Graph

A weight or cost is assigned to each edge

- In adjacency matrix, we can incorporate this weight and use ∞ to represent the case of no linking edge
- In adjacency linked list, add a data item to store the weight

Applications:

- Traveling salesman
- Shortest path etc.



(a)

| | <i>a</i> | <i>b</i> | <i>c</i> | <i>d</i> |
|----------|----------|----------|----------|----------|
| <i>a</i> | ∞ | 5 | 1 | ∞ |
| <i>b</i> | 5 | ∞ | 7 | 4 |
| <i>c</i> | 1 | 7 | ∞ | 2 |
| <i>d</i> | ∞ | 4 | 2 | ∞ |

(b)

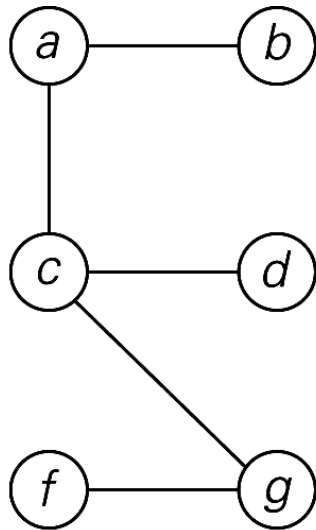
| | |
|----------|--|
| <i>a</i> | → <i>b</i> , 5 → <i>c</i> , 1 |
| <i>b</i> | → <i>a</i> , 5 → <i>c</i> , 7 → <i>d</i> , 4 |
| <i>c</i> | → <i>a</i> , 1 → <i>b</i> , 7 → <i>d</i> , 2 |
| <i>d</i> | → <i>b</i> , 4 → <i>c</i> , 2 |

(c)

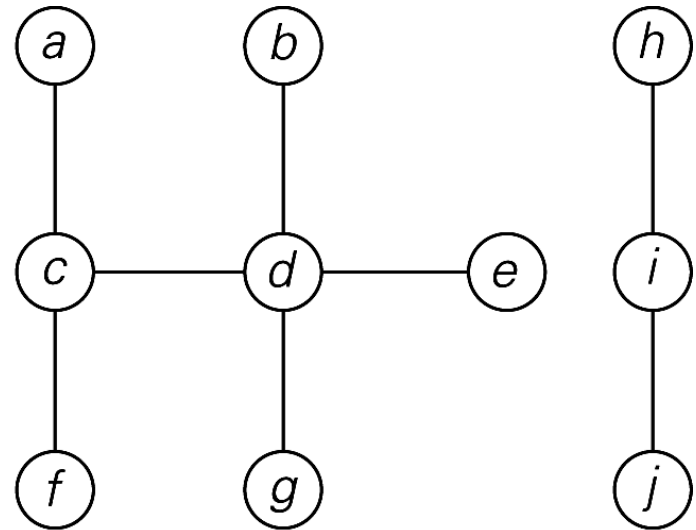
FIGURE 1.8 (a) Weighted graph. (b) Its weight matrix. (c) Its adjacency lists.

Special Graphs -- Trees

- A connected acyclic graph
- An unconnected acyclic graph → a forest



(a)



(b)

FIGURE 1.10 (a) Tree. (b) Forest.

Trees

- One simple path between vertices in a tree exists and is unique
- For a tree we have $|E|=|V|-1$
 - A sufficient and necessary condition
 - Why?
- Rooted tree → root at level 0

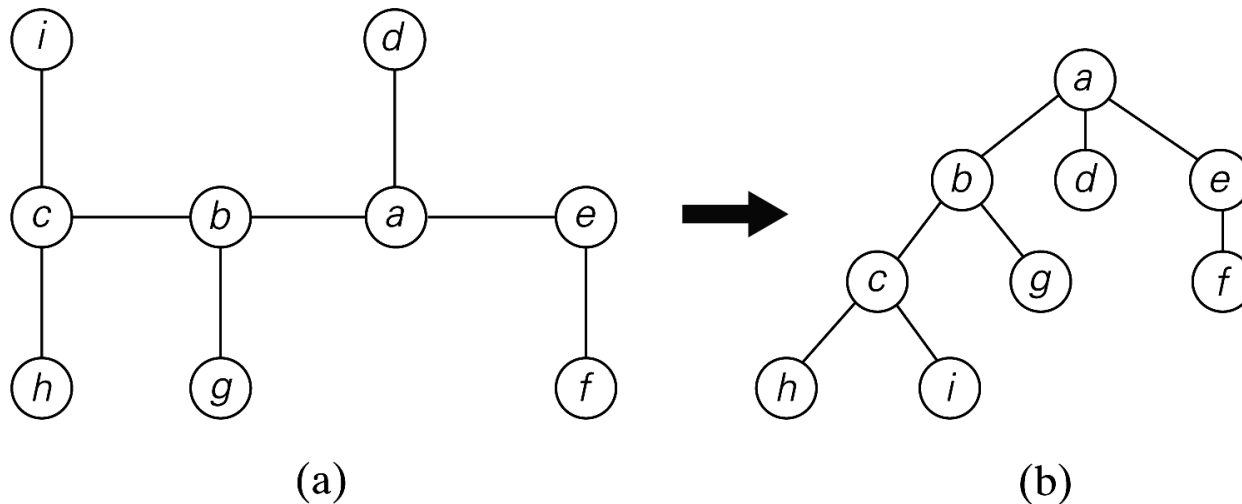
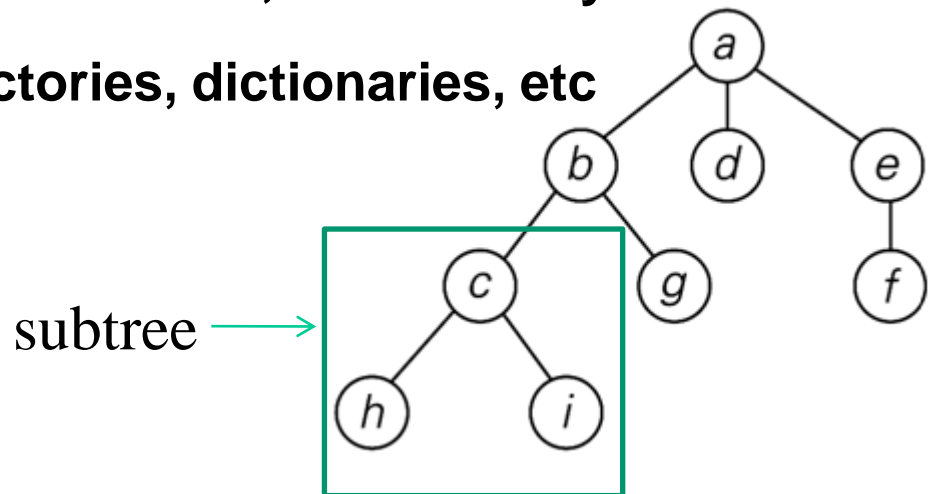


FIGURE 1.11 (a) Free tree. (b) Its transformation into a rooted tree.

Rooted Trees

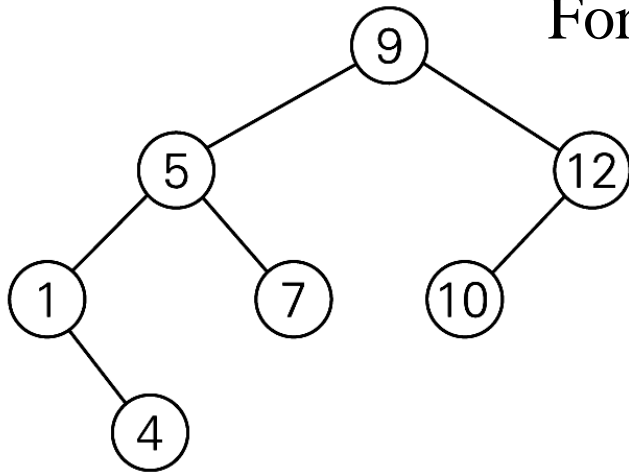
- **Some concepts related to a vertex in a rooted tree**
 - Root/leaf
 - (Proper) Ancestors/descendants
 - Parent/Child/Siblings
 - Depth of a vertex v : the length of the simple path $root \rightarrow v$
 - Height of a tree: the longest simple path in the tree
- **Operations: insert/delete a node, search a key**
- **Applications: file directories, dictionaries, etc**
- **Ordered rooted trees**
 - Binary trees



Binary Tree

Each vertex has no more than two children

➤ **The height of a binary tree is** the length of the longest simple path from the root to the leaf



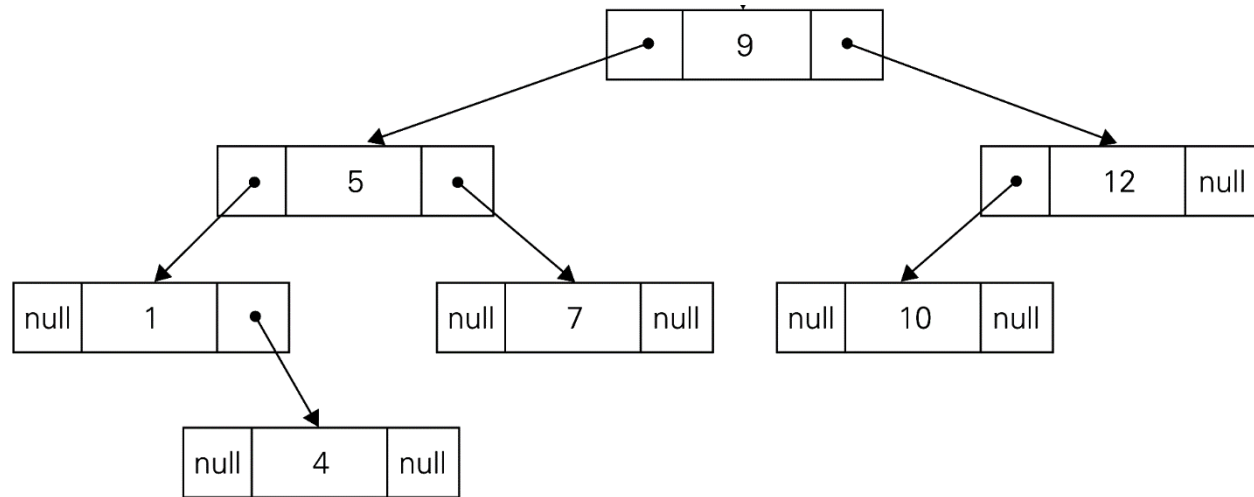
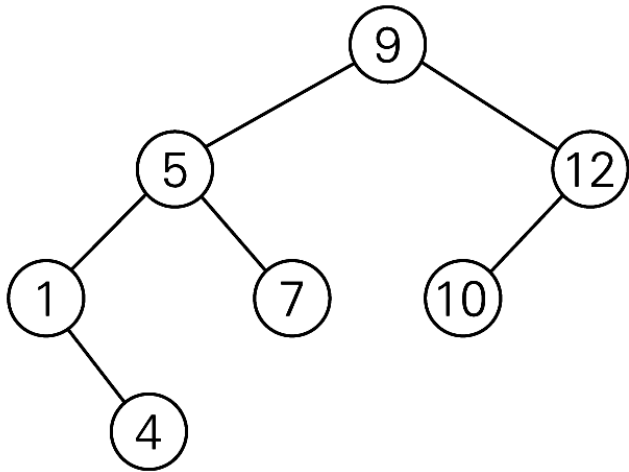
For a binary tree with n nodes, its height h satisfies

$$\lfloor \log_2 n \rfloor \leq h \leq n - 1$$

A complete tree

A linked list

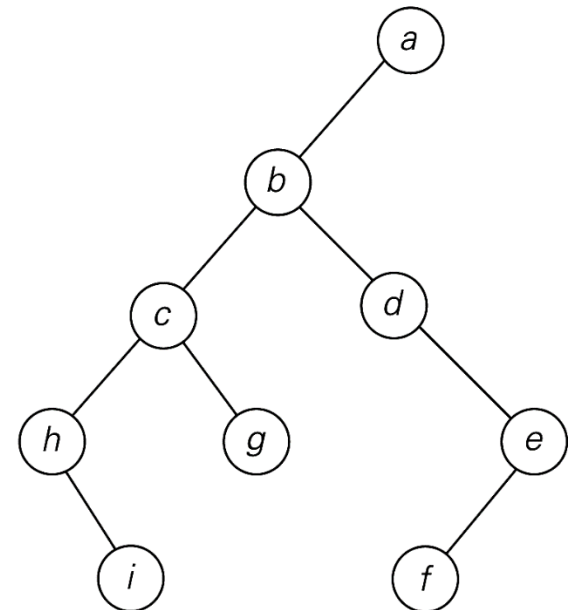
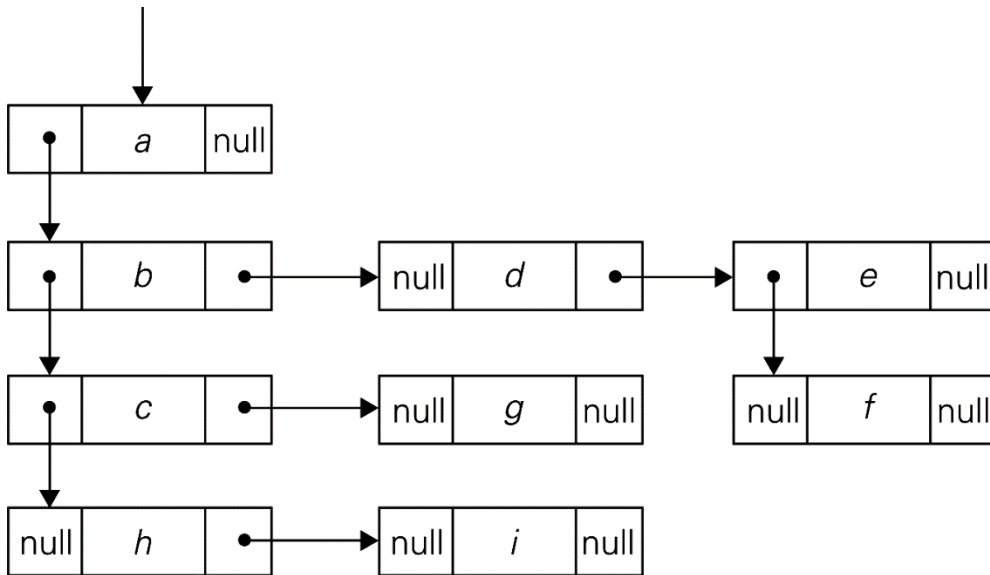
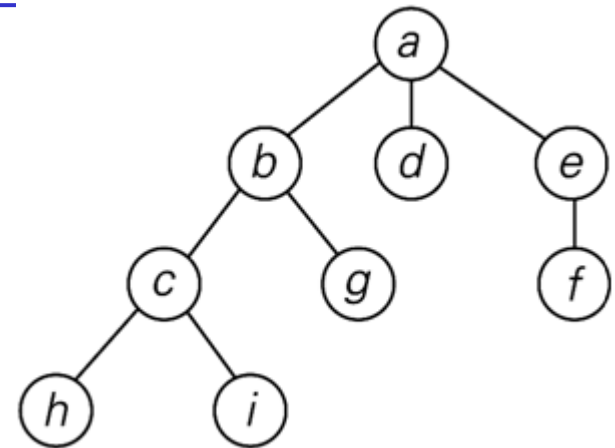
Implementation of a Binary Tree



Transform an Ordered Tree to a Binary Tree

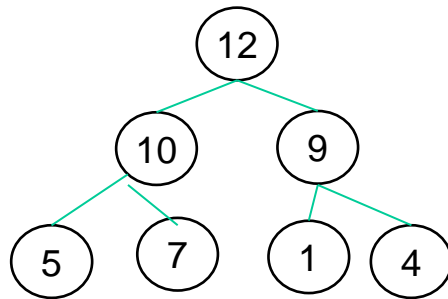
First child-next sibling

- Left child: first child
- Right child: next sibling

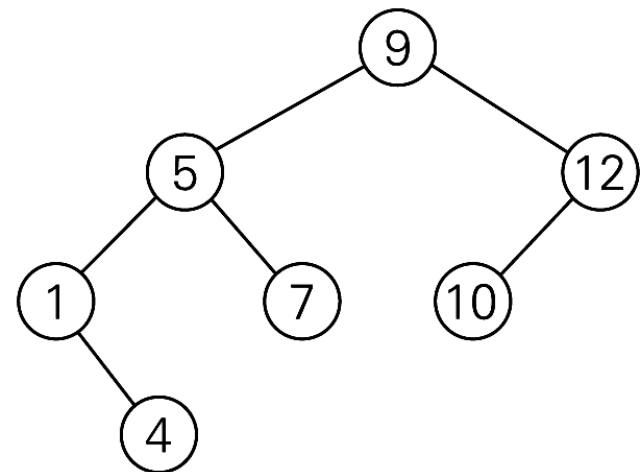


Special Binary Trees

- **Heap (priority queue)**
- **Binary search tree** – the number associated with the parent is larger than the ones in its left subtree and smaller than the ones in its right subtree.



heap



Binary search tree

Sets and Dictionaries

Set:

- Unordered collection of unique items
- Set representations: bit vector or a list
 - Difference between a set and a list (ordered and not unique)
- Set operations
 - Membership checking
 - Set union & set intersection

Dictionary: a set with the operations of searching, adding, and deleting

Theoretical Analysis of Algorithm -- Algorithm Efficiency

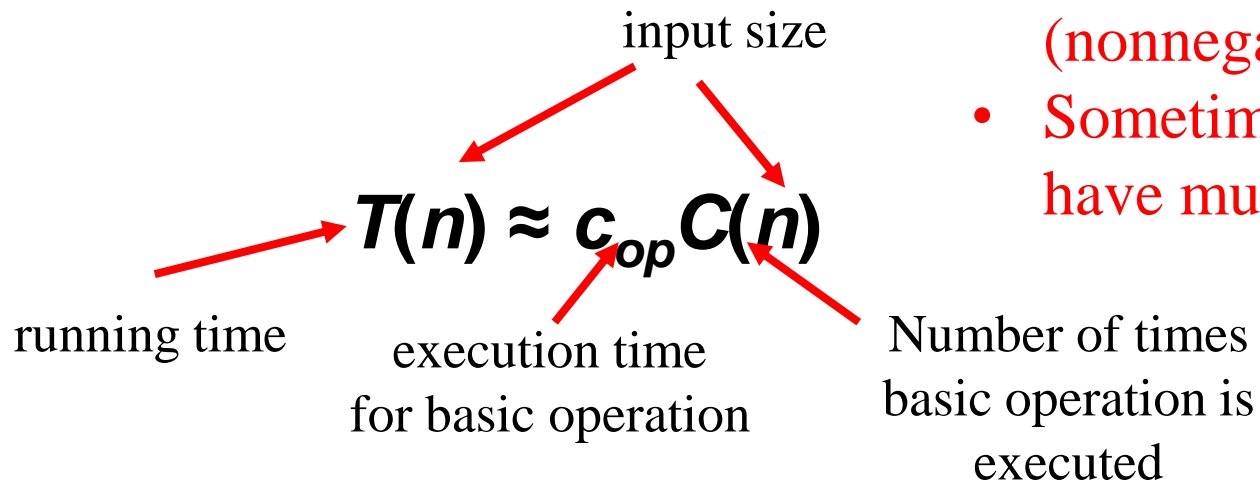
Time efficiency

Space efficiency

Theoretical Analysis of Time Efficiency

Time efficiency is analyzed by determining the number of repetitions of the basic operation as a function of input size

Basic operation: the operation that contributes most towards the running time of the algorithm.




- n is a natural number (nonnegative integer)
- Sometimes, we can have multiple numbers.

Why not use unit of time?

How to Choose Basic Operations


Basic operation should be simple and cannot be represented by other operations in the same algorithm

$$T(n) = T_1(n) + \dots + T_M(n) = c_{op,1}t_1(n) + \dots + c_{op,M}t_M(n)$$

If $t_1(n) \approx t_2(n)$, $c_{op,1} \gg c_{op,2}$  Operation 1 is the basic operation.

For example, choose “*” or “/” rather than “+” and “-”

http://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations

If $t_1(n) \ll t_2(n)$, $c_{op,1} \approx c_{op,2}$  Operation 2 is the basic operation

Example: Selection Sort

ALGORITHM *SelectionSort*($A[0..n - 1]$)
//Sorts a given array by selection sort
//Input: An array $A[0..n - 1]$ of orderable elements
//Output: Array $A[0..n - 1]$ sorted in ascending order
for $i \leftarrow 0$ **to** $n - 2$ **do**
 $min \leftarrow i$
 for $j \leftarrow i + 1$ **to** $n - 1$ **do**
 if $A[j] < A[min]$ $min \leftarrow j$
 swap $A[i]$ and $A[min]$

Input size? Basic operation?

Input Size and Basic Operation Examples

| <i>Problem</i> | <i>Input size measure</i> | <i>Basic operation</i> |
|---|----------------------------------|---|
| Search for key in a list of n items | # of items in list -- n | Key comparison |
| Multiply two matrices of floating point numbers | Dimensions of matrices | Floating point multiplication |
| Compute a^n | n or # of bits in n | Floating point multiplication |
| Graph problem | # of vertices and/or # of edges | Visiting a vertex or traversing an edge |

Types of Formulas for Counting Basic Operations

Exact formula

e.g., $C(n) = n(n-1)/2$

Formula indicating order of growth with specific multiplicative constant

e.g., $C(n) \approx 0.5 n^2$

Formula indicating order of growth with unknown multiplicative constant

e.g., $C(n) \approx cn^2$

Order of Growth

The number of operations grows with an increase of the input size.

- Example: assume $T(n) \approx c_{op}n^2$
 - How much faster will algorithm run on computer that is twice as fast?
 - How much longer does it take to solve problem of double input size?

Order of growth determines the growth rate as the input size goes to infinity

- Recall the example of computing GCD:
 - Euclid's algorithm: $5 \log_{10} n$
 - Algorithm 2: n
 - Algorithm 3: $cn \log \log n$

| n | 1 | 5 | 10 | 100 | 1,000 | 10,000 |
|----------|-----|-----|----|-----|-------|--------|
| Euclid's | 0 | 3.5 | 5 | 10 | 15 | 20 |
| Al. 2 | 1 | 5 | 10 | 100 | 1,000 | 10,000 |
| Al. 3 | N/A | 2.4 | 8 | 153 | 1,933 | 22,203 |

More Examples of Order of Growth

| n | $c(n)$ | | | | | | |
|--------|------------|--------|------------------|-----------|-----------|---------------------|----------------------|
| | $\log_2 n$ | n | $n \log_2 n$ | n^2 | n^3 | 2^n | $n!$ |
| 10 | 3.3 | 10^1 | $3.3 \cdot 10^1$ | 10^2 | 10^3 | 10^3 | $3.6 \cdot 10^6$ |
| 10^2 | 6.6 | 10^2 | $6.6 \cdot 10^2$ | 10^4 | 10^6 | $1.3 \cdot 10^{30}$ | $9.3 \cdot 10^{157}$ |
| 10^3 | 10 | 10^3 | $1.0 \cdot 10^4$ | 10^6 | 10^9 | | |
| 10^4 | 13 | 10^4 | $1.3 \cdot 10^5$ | 10^8 | 10^{12} | | |
| 10^5 | 17 | 10^5 | $1.7 \cdot 10^6$ | 10^{10} | 10^{15} | | |
| 10^6 | 20 | 10^6 | $2.0 \cdot 10^7$ | 10^{12} | 10^{18} | | |



Time complexity increases!

Best-case, Average-case, Worst-case

For some algorithms efficiency depends on the property of input:

Worst case: $W(n)$ – max # of basic operations over inputs of size n

Best case: $B(n)$ – min # of basic operations over inputs of size n

Average case: $A(n)$ – “average” # of basic operations over inputs of size n

- # of the basic operation will be executed on typical input
- NOT the average of worst and best case
- Expected number of basic operations repetitions considered as a random variable under some assumption about the probability distribution of all possible inputs of size n

Example: Sequential Search

Problem: Given a list of n elements and a search key K , find an element equal to K , if any.

Algorithm: Scan the list and compare its successive elements with K until either a matching element is found (*successful search*) or the list is exhausted (*unsuccessful search*)

Worst case

$$C_{worst}(n) = n$$

Best case

$$C_{best}(n) = 1$$

Average case

$$C_{average}(n) = ?$$

$$C_{average}(n) = 1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \dots + n \cdot \frac{p}{n} + n \cdot (1-p) = n(1 - \frac{p}{2}) + \frac{p}{2}$$

Next Time

Fundamentals of the analysis of algorithm efficiency

- Analysis framework
- Big O notation, big theta notation, and big omega notation

Reading Assignments: Chapter 2