Computer Algorithms

Lecture 1: Introduction – Ch 1

Dr. Manal Helal, Spring 2014.

http://moodle.manalhelal.com

Course Description

This course introduces students to the analysis and design of computer algorithms. Topics covered include problem types, algorithm performance analysis and limitations, algorithm design techniques such as: brute force, decrease and conquer, divide and conquer, Transform and conquer, dynamic programming, greedy techniques, and iterative techniques.

Course learning objectives

1 Analysis: The ability to analyse algorithmic efficiency via different Design Techniques

2 Design: The ability to create algorithms via different design techniques

Course Outline

1. Introduction – Ch 1
2. Fundamentals of the Analysis of Algorithm Efficiency – Ch2
3. Brute Force and Exhaustive Search – Ch3
4. Decrease-and-Conquer – Ch4
5. Divide-and-Conquer – Ch5
6. Transform-and-Conquer – Ch6
7. Midterm
8. Space and Time Trade-Offs – Ch7
9. Dynamic Programming – Ch8
10. Greedy Technique – Ch9
11. Iterative Improvement – Ch10
12. Limitations of Algorithm Power – Ch11
13. Coping with the Limitations of Algorithm Power – Ch12
14. Revision
15. Presentations

Grading Scheme

20% - Midterm Exam – Week 7

15% - Lecture & Lab Quizzes

5% Section Submissions

10% - Assignments

10% - Case Study– Week 12

40% - Final Exam – will be announced

Grading scale

A+ =	95%, ∝)	A =	[90%, 95%)
A- =	[85%, 90%)	B+ =	[80%, 85%)
B =	[75%, 80%)	B- =	[70%, 75%)
C=	[65%, 70%)	C =	[60%, 65%)
C-=	[55%, 60%)	D =	[50%, 55%)

Textbook & References

Anany Levitin, Introduction to the Design and Analysis of Algoritms, 3rd Edition, Pearson, 2012.

References

Steven S. Skiena. The Algorithm Design Manual. Springer-Verlag, 1998

Rules

All communications through: http://moodle.manalhelal.com/course/view.php?id=6, please subscribe today!

Attendance

Attendance is a must in all CCIT courses. A consecutive 3 absences will result in course forced withdrawal. 2 sections/labs count for 1 lecture. For example, a student absent for 2 lectures & 3 labs will be withdrawn. Medical and other excuses should be submitted to the department.

Submissions

Assignments and all graded activities are given codes, such as: ass1, ass2, proj1, exer1, … etc, and announced allowed submission file extensions, and due dates. All submissions should be done electronically using moodle website. Files submitted should be named "code_StudentID.ext", where code is the graded activity code, StudentID is your numerical AASTMT student ID, and ext is the announced allowed extension for each graded activity. If assignment 1 is coded as "ass1" and the allowed file extension is pdf, and your ID is 111238090, then the submitted file name should be: "ass1_111238090.pdf". Due dates are final and there is a 10% reduction in the earned grade for each late day after the due date. After 5 days of the due date, no submissions are accepted, and model answer will be published on the website.

Academic Honesty

First academic honesty preaching will result in a disciplinary action ranging from zero mark in the graded activity and up to forced course withdrawal or forced academy dismissal, as regulated by AASTMT policies. This includes copied assignments/projects, exam cheating of all types, inadequate individual participation in teamwork – more on course Description Document, and College and Academy Hand-Books.

Studying Plan & Teaching Method

•Every lecture is followed with exercise problems to be attempted in the Section and uploaded on moodle. 5% of the total course marks are for these section submissions. The section exercises will help deepen your understanding and students are expected to do their best to attempt them independently.

•There are 4 assignments that will vary from analysis, design and programming requirements.

•Asking questions are encouraged in this preference order:
  - •In Moodle to have the question and the answer available to everyone in written form to get back to while studying.
  - •In office hours,
  - •then finally in lecture and section times to avoid lengthy interruptions and delay in course contents.
  - •Please don't accumulate material without full understanding and use the lecturer and the TA as much as you can to do your best.

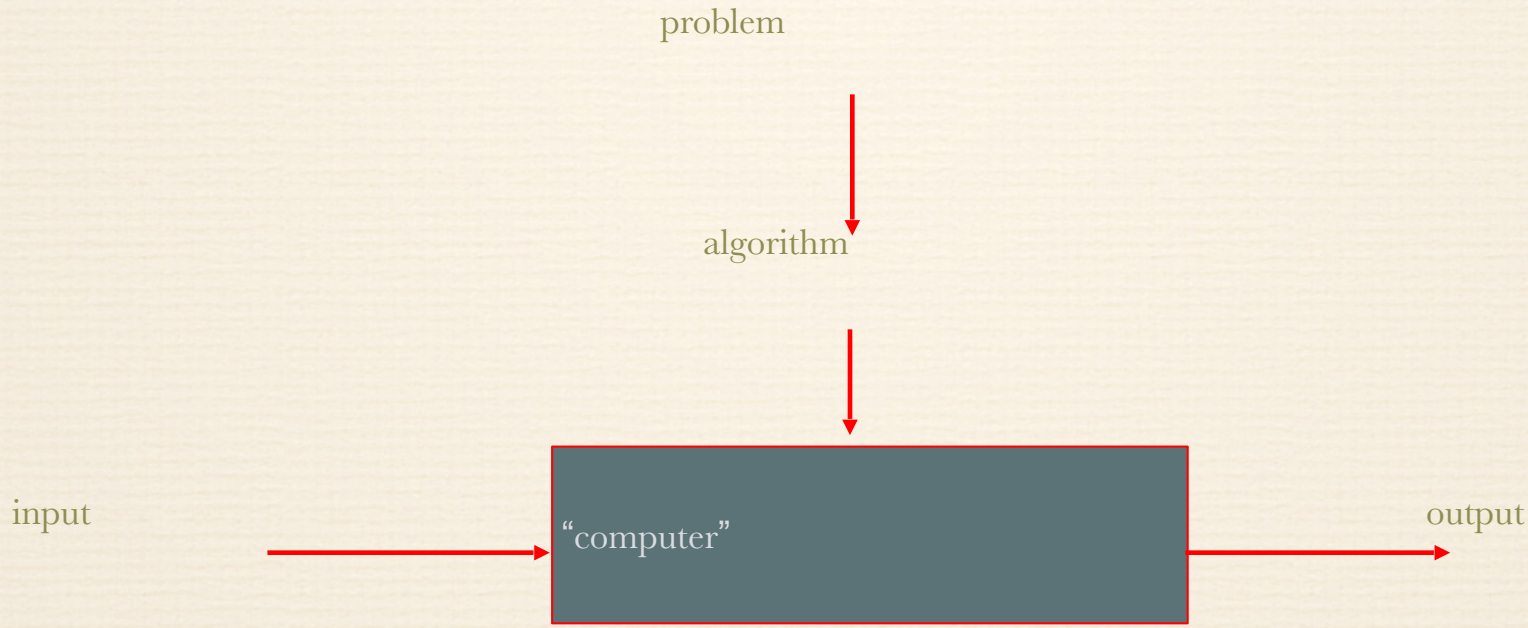
•Understanding theoretical concepts in lectures, attempting and submitting all section problems, doing all assignments, and engaging in a good case study, are the methods to study for the lecture/lab quizzes, midterm and final exams.

Lecture Learning Objectives

1. Define What "Algorithm" means.
2. Understand the Fundamentals of Algorithmic Problem Solving.
3. Know Important Problem Types
4. Revise Fundamental Data Structures.

What is an 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.



Greatest Common Divisor Algorithms

We will study 3 algorithms for GCD computation to illustrate the following:

- The non-ambiguity requirement for each step of an algorithm cannot be com- promised.
- The range of inputs for which an algorithm works has to be specified carefully.
- The same algorithm can be represented in several different ways.
- There may exist several algorithms for solving the same problem.
- Algorithms for the same problem can be based on very different ideas and can solve the problem with dramatically different speeds.

Alg 1: Euclid's Algorithm

Problem: Find gcd(m,n), the greatest common divisor of two nonnegative, not both zero integers m and n

Examples: gcd(60,24) = 12, gcd(60,0) = 60, gcd(0,0) = ?

Euclid's algorithm is based on repeated application of equality

$$\gcd(m,n) = \gcd(n, m \bmod n)$$

until the second number becomes 0, which makes the problem
trivial.

Example: gcd(60,24) = gcd(24,12) = gcd(12,0) = 12

Two descriptions of Euclid's algorithm

Step 1 If $n = 0$, return m and stop; otherwise go to Step 2
Step 2 Divide m by n and assign the value of the remainder to r
Step 3 Assign the value of n to m and the value of r to n. Go to Step 1.

ALGORITHM Euclid(m, n)
//Computes gcd(m, n) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while $n \neq 0$ do
    $r \leftarrow m \bmod n$
    $m \leftarrow n$
    $n \leftarrow r$
return m

Alg 2: Minimum Iterating down to Zero

Consecutive integer checking algorithm

Step 1 Assign the value of $\min\{m,n\}$ to t

Step 2 Divide m by t. If the remainder is 0, go to Step 3; otherwise, go to Step 4

Step 3 Divide n by t. If the remainder is 0, return t and stop; otherwise, go to Step 4

Step 4 Decrease t by 1 and go to Step 2

Alg 3: Using Primes

Middle-school procedure:

Step 1 Find the prime factorization of m

Step 2 Find the prime factorization of n

Step 3 Find all the common prime factors

Step 4 Compute the product of all the common prime factors and return it as gcd(m,n)

Is this an algorithm?

60 = 2 . 2 . 3 . 5

24 = 2 . 2 . 2 . 3

gcd(60, 24) = 2 . 2 . 3 = 12.

Sieve of Eratosthenes

n = 25

p	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
2	2	3		5		7		9		11		13		15		17		19		21		23		25
3	2	3		5		7				11		13				17		19				23		25
5	2	3		5		7				11		13				17		19				23		

p.p < n
P <= √n

Sieve of Eratosthenes

Input: Integer $n \geq 2$

Output: List of primes less than or equal to n

for $p \leftarrow 2$ to n do $A[p] \leftarrow p$

for $p \leftarrow 2$ to $\lfloor \sqrt{n} \rfloor$ do

    if $A[p] \neq 0$ //p hasn't been previously eliminated from the list

    $j \leftarrow p * p$

    while $j \leq n$ do

        $A[j] \leftarrow 0$ //mark element as eliminated

        $j \leftarrow j + p$

//copy the remaining elements of A to array L of the primes

i←0

for p ← 2 **to** n **do**

    if A[p] ≠ 0

        L[i]←A[p]

        i←i+1

return L

Example: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

EXCERCISE

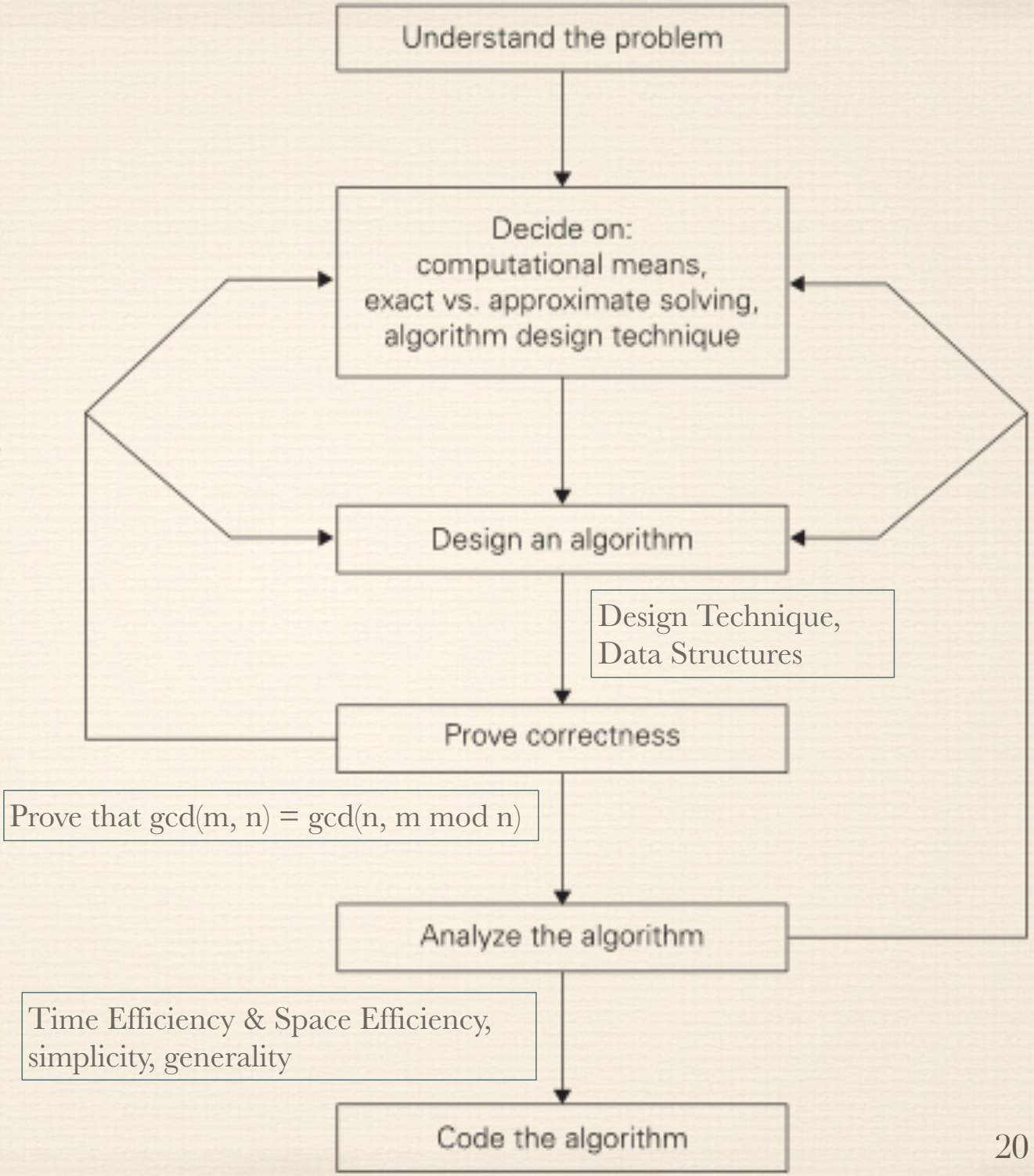
Exercises 1.1:

1. Do some research on al-Khorezmi (also al-Khwarizmi), the man from whose name the word "algorithm" is derived. In particular, you should learn what the origins of the words "algorithm" and "algebra" have in common.

Why study algorithms?

- We can consider algorithms to be procedural solutions to problems.
- These solutions are not answers but specific instructions for getting answers.
- Theoretical importance
  - the core of computer science
- Practical importance
  - A practitioner's toolkit of known algorithms
  - Framework for designing and analyzing algorithms for new problems

Algorithm Design & Analysis Steps



Two main issues related to algorithms

1. How to design algorithms

2. How to analyze algorithm efficiency
   - How good is the algorithm?
     * time efficiency & space efficiency
   - Does there exist a better algorithm?
     * lower bounds & optimality

Algorithm Design Techniques/Strategies

Brute force	Greedy approach
Divide and conquer	Dynamic programming
Decrease and conquer	Iterative improvement
Transform and conquer	Backtracking
Space and time trade-offs	Branch and bound

EXCERCISE

Exercises 1.2:

2. New World puzzle There are four people who want to cross a rickety bridge; they all begin on the same side. You have 17 minutes to get them all across to the other side. It is night, and they have one flashlight. A maximum of two people can cross the bridge at one time. Any party that crosses, either one or two people, must have the flashlight with them. The flashlight must be walked back and forth; it cannot be thrown, for example. Person 1 takes 1 minute to cross the bridge, person 2 takes 2 minutes, person 3 takes 5 minutes, and person 4 takes 10 minutes. A pair must walk together at the rate of the slower person's pace.

Important Problem Types - I

- Sorting
  - Key, numeric or alphabetical
  - Stable and in-place algorithms
  - Best Algorithm is using n $\log_2$ n comparisons
- Searching
  - Search Key
  - Sequential or Binary search algorithms
- String Processing
  - String matching
- Graph Problems
  - Traveling Salesman Problem (TSP), Graph-Colouring Problem
  - Applications: transportation, communication, social and economic networks, project scheduling, and games

Important Problem Types - II

- Combinatorial Problems
  - Grow fast with problem size.
  - No known algorithms as believe, with no proof.
  - The shortest-path problem is an exception combinatorial problem with known algorithms.
- Geometric Problems
  - Points, Lines, and Polygons.
  - Applications: Computer Graphics, Robotics, and Tomography.
  - Problems: closest-pair, convex-hull
- Numerical Problems
  - Mathematical objects of continuous nature: solving equations and systems of equations, computing definite integrals, evaluating functions, … etc.
  - Approximation because of real numbers causing an accumulation of round-off error.
  - Applications domains: Scientific and Engineering applications, and now business applications.

EXCERCISE

Exercises 1.3:

1. Consider the algorithm for the sorting problem that sorts an array by counting, for each of its elements, the number of smaller elements and then uses this information to put the element in its appropriate position in the sorted array:

```
ALGORITHM ComparisonCountingSort(A[0..n − 1])
//Sorts an array by comparison counting
//Input: Array A[0..n − 1] of orderable values
//Output: Array S[0..n − 1] of A's elements sorted
// in nondecreasing order
    for i ← 0 to n − 1 do
        Count[i]←0
    for i ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
                if A[i] < A[j]
                        Count[j]←Count[j]+1
                else
                        Count[i] ← Count[i] + 1
    for i ← 0 to n − 1 do
        S[Count[i]]←A[i]
    return S
```

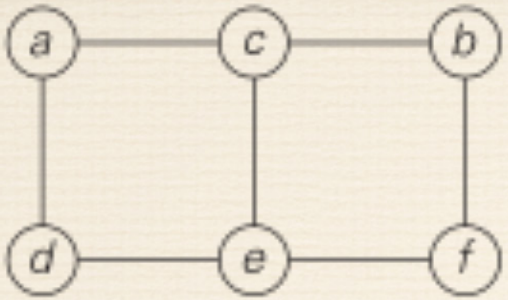
a. Apply this algorithm to sorting the list:
60, 35, 81, 98, 14, 47.

b. Is this algorithm stable?
c. Is it in-place?

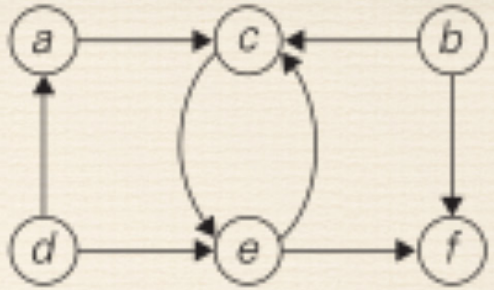
Fundamental Data Structures

Linear Data Structures	List, Array, String, Linked List
Linear Abstract Data Structures	Stack, Queue, Priority Queue, Sets
Non-Linear Data Structures	Graph, Tree, Dictionary or Tuples

Graphs



V ={a,b,c,d,e,f}
E={(a,c),(a,d),(b,c),(b,f),(c,e),(d,e),(e,f)}

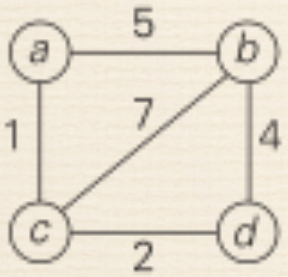


V ={a,b,c,d,e,f}
E={(a,c),(b,c),(b,f),(c,e),(d,a),(d,e),(e,c),(e,f)}

	a	b	c	d	e	f
a	0	0	1	1	0	0
b	0	0	1	0	0	1
c	1	1	0	0	1	0
d	1	0	0	0	1	0
e	0	0	1	1	0	1
f	0	1	0	0	1	0

a	→	c	→	d	
b	→	c	→	f	
c	→	a	→	b	→ e
d	→	a	→	e	
e	→	c	→	d	→ f
f	→	b	→	e	

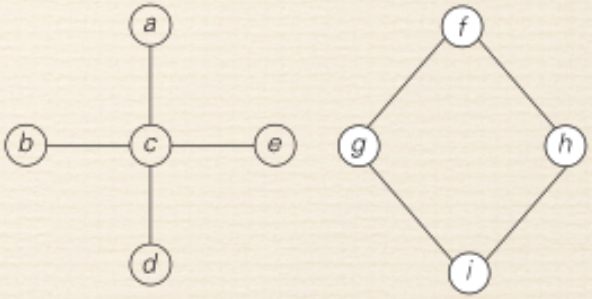
FIGURE 1.7 (a) Adjacency matrix and (b) adjacency lists of the graph in Figure 1.6a.



	a	b	c	d
a	∞	5	1	∞
b	5	∞	7	4
c	1	7	∞	2
d	∞	4	2	∞

a	→ b, 5 → c, 1
b	→ a, 5 → c, 7 → d, 4
c	→ a, 1 → b, 7 → d, 2
d	→ b, 4 → c, 2

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

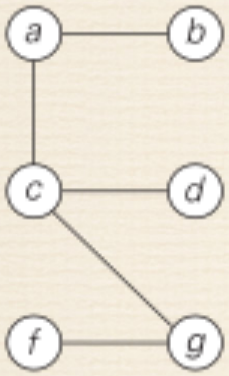


Graph that is not connected.

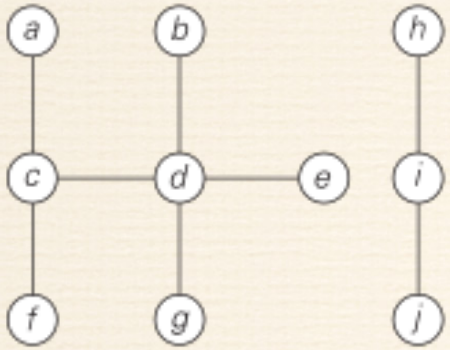
Trees (Connected Acyclic Graph)

$|E| = |V| - 1$

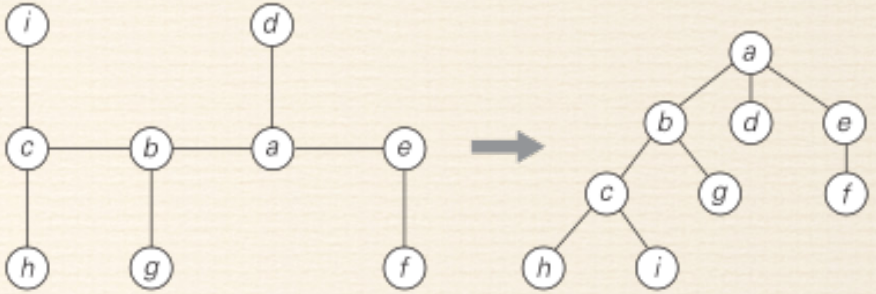
$\lfloor \log_2 n \rfloor \leq h \leq n - 1$



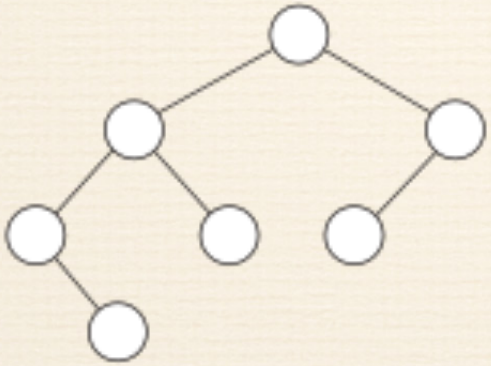
Tree



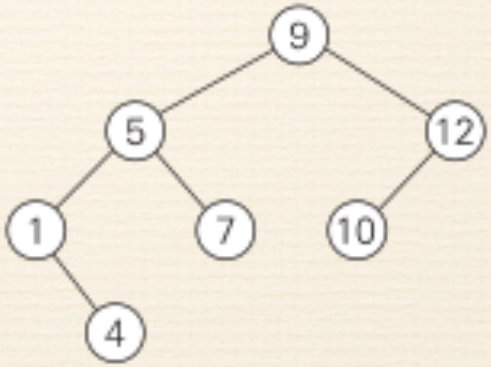
Forest



Free tree transformation into a rooted tree.

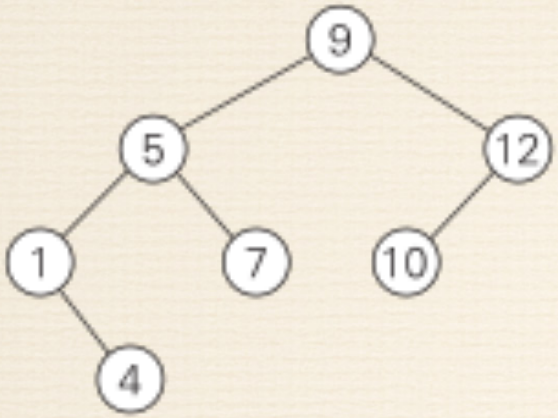


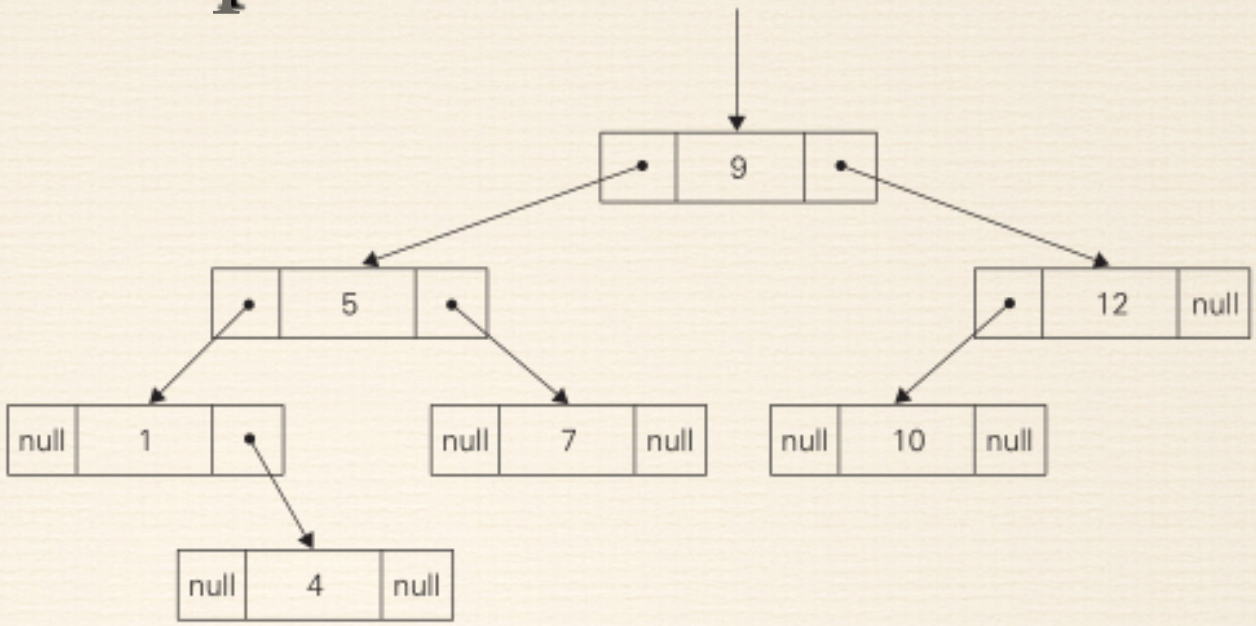
Binary tree.



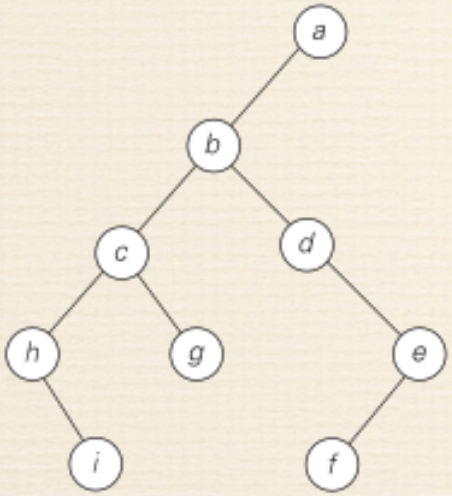
Binary search tree

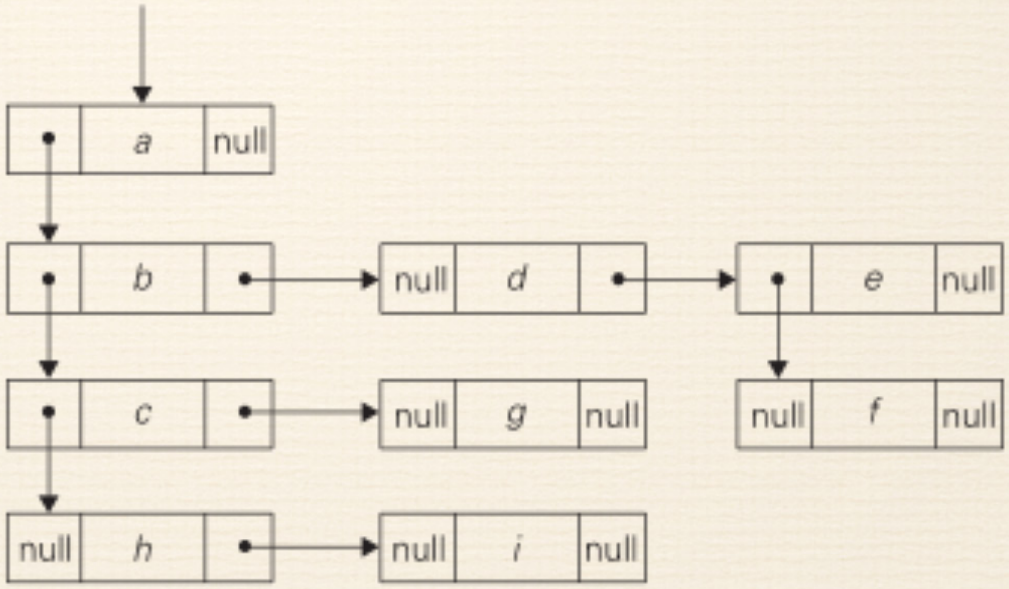
Tree Representation





Standard implementation of the binary search tree





First child–next sibling representation of the tree

Sets

- Notation: $A=\{x \mid x \in N, x \bmod 3 = 1\}$
  $N = \{1,2,3,...\}$
- Union: $A \cup B$
- Intersection: $A \cap B$
- Complement: $\bar{A}$
- Cardinality: $|A|$
- Cartesian Product:
  $A \times B = \{ (x,y) \mid x \in A \text{ and } y \in B\}$
- Set Representation: bit vector

Any order, and repetition is alright in sets, but not in sequences or tuples

Ex: $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and $S \subset U = \{2, 3, 5, 7\}$, then
S is represented by bit String = 011010100

Sets Examples

- $L_{<6} = \{ x \mid x \in N , x<6 \}$
- $L_{prime} = \{x \mid x \in N, x \text{ is prime}\}$
- $L_{<6} \cap L_{prime} = \{2,3,5\}$
- $\sum = \{0,1\}$
- $\sum \times \sum = \{(0,0), (0,1), (1,0), (1,1)\}$
- Formal: $A \cap B = \{ x \mid x \in A \text{ and } x \in B\}$

Power Set

- "Set of all subsets"
- Formal: $P(A) = \{ S \mid S \subseteq A \}$
- Example: $A = \{x,y\}$
- $P(A) = \{ \{\} , \{x\} , \{y\} , \{x,y\} \}$
- Power set
- Note the different sizes: for finite sets
- $|P(A)| = 2^{|A|}$
- $|A \times A| = |A|^2$

Dictionaries

- A set that implement the functions: search/add/delete items and require balance between the efficiency of the three operations.
- Represented as arrays, linked lists, hash tables, and balanced search trees.
- Set union problem is partitioning n-element set into a collection of disjoint subsets.

Next

Read Chapter 2 “Fundamentals of the Analysis of Algorithm Efficiency” to prepare for the discussions in lecture.