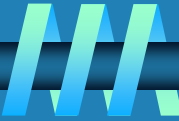
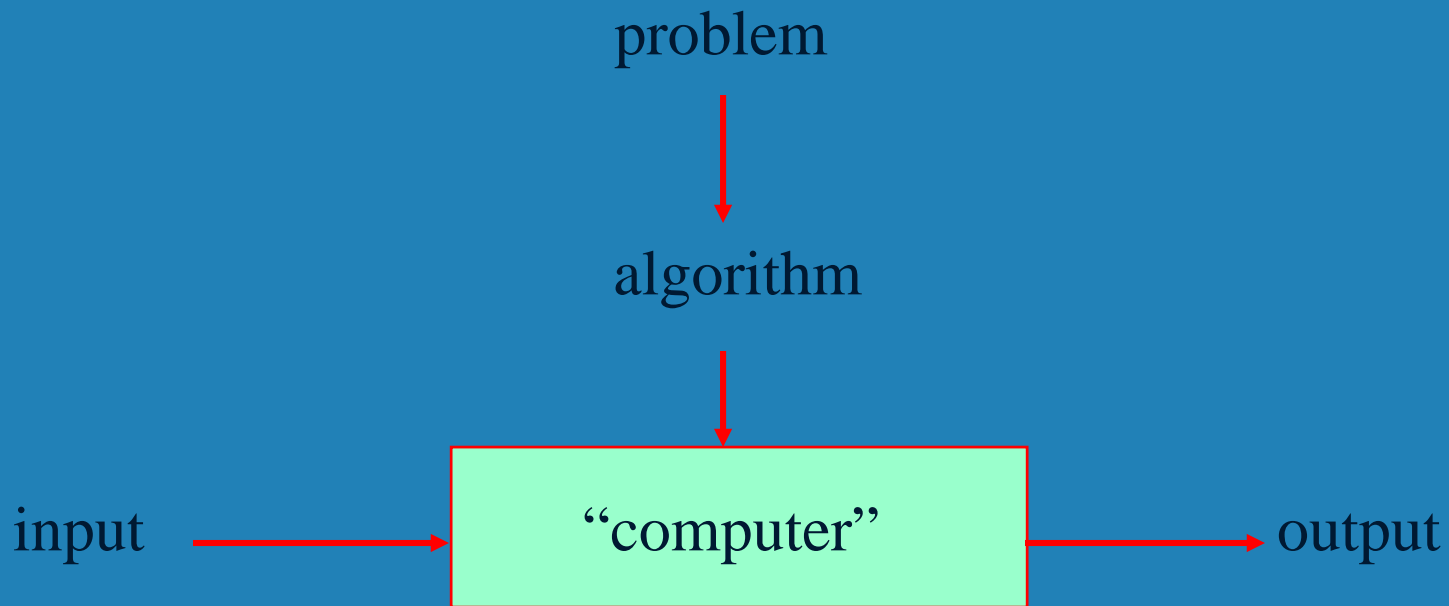


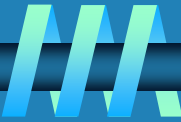
What is an algorithm?



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.



Euclid's Algorithm



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

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

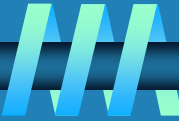
Euclid's algorithm is based on repeated application of equality

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

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 for the remainder to r

Step 3 Assign the value of n to m and the value of r to n . Go to Step 1.

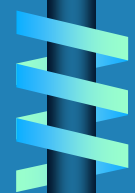
while $n \neq 0$ **do**

$r \leftarrow m \bmod n$

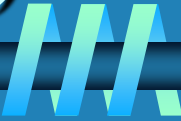
$m \leftarrow n$

$n \leftarrow r$

return m



Other methods for computing $\gcd(m,n)$



Consecutive integer checking algorithm

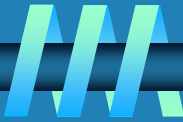
Step 1 Assign the value of $\min\{m,n\}$ to t

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

Other methods for $\gcd(m,n)$ [cont.]



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?

Sieve of Eratosthenes

Algorithm for generating consecutive primes not exceeding any given integer $n > 1$.

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*

$j \leftarrow p * p$ *//all smaller multiples $2p, \dots, (p-1)p$ are eliminated*

while $j \leq n$ **do**

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

$j \leftarrow j + p$ *// $p*(p+1), p*(p+2), \dots$*

Sieve of Eratosthenes Continues

//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

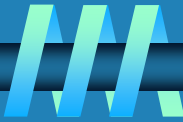
Sieve of Eratosthenes Continues

As an example, consider the application of the algorithm to finding the list of primes not exceeding $n=25$:

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	3		5		7		9		11		13		15		17		19		21		23		25
2	3		5		7				11		13				17		19				23		25
2	3		5		7				11		13				17		19				23		

For this example, no more passes are needed because they would eliminate numbers already eliminated on previous iterations of the algorithm. The remaining numbers on the list are the consecutive primes less than or equal to 25.

Why study algorithms?



⌘ 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

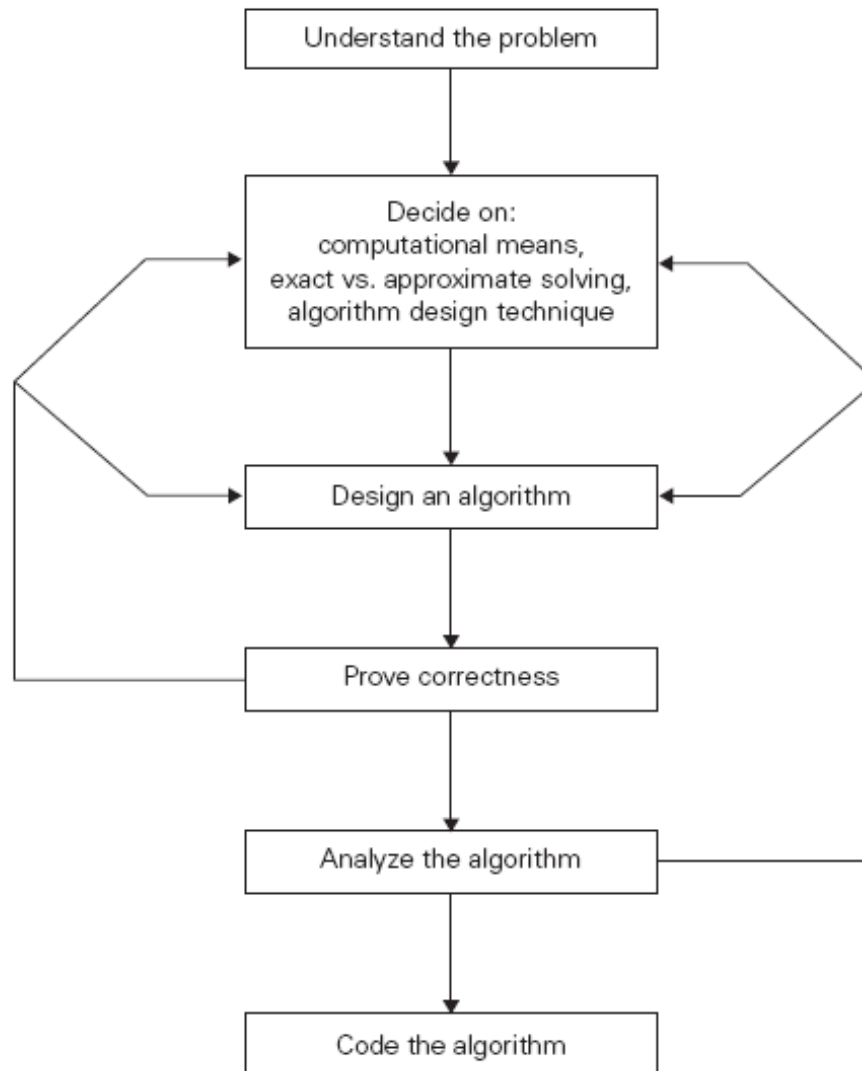
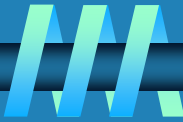
Two main issues related to algorithms




⌘ How to design algorithms

⌘ How to analyze algorithm efficiency

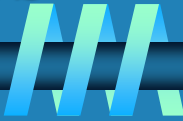
Design and Analysis of Algorithm



 Algorithm design and analysis process.

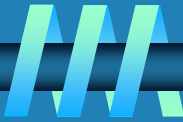


Algorithm design techniques/strategies



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

Analysis of algorithms



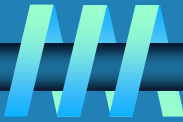
❧ How good is the algorithm?

- time efficiency
- space efficiency

❧ Does there exist a better algorithm?

- lower bounds
- optimality

Important problem types



∩ sorting

Two properties: stable and in-place

∩ searching

∩ string processing

∩ graph problems

Traveling salesman problem, graph-coloring problem

∩ combinatorial problems

Finding a combinatorial object that satisfies certain constraints

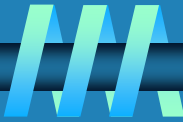
∩ geometric problems

Closest-pair problem, convex-hull problem

∩ numerical problems

∩ *Example: solving equations, evaluating functions, computing definite integrals and so on*

Fundamental data structures



∩ list

- array
- linked list
- string

∩ stack

LIFO

∩ queue

FIFO

∩ priority queue

HEAP

∩ Graph

$G = \langle V, E \rangle$

Representation:

Adjacency matrix

Adjacency list

∩ Tree (*connected acyclic graph*)

∩ set and dictionary

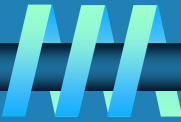


FIGURE 1.3 Array of n elements.

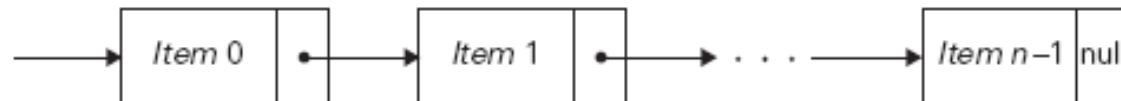


FIGURE 1.4 Singly linked list of n elements.

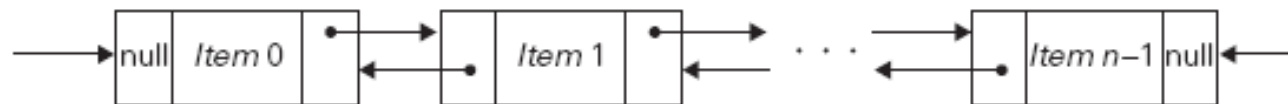
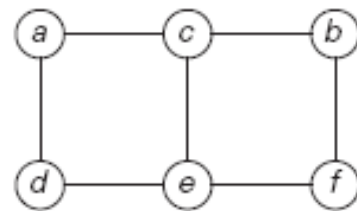
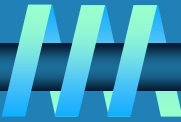
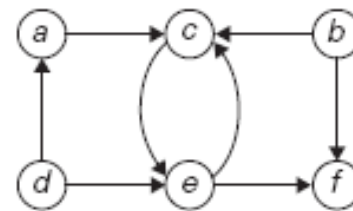


FIGURE 1.5 Doubly linked list of n elements.



(a)



(b)

FIGURE 1.6 (a) Undirected graph. (b) Digraph.

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
<i>a</i>	0	0	1	1	0	0
<i>b</i>	0	0	1	0	0	1
<i>c</i>	1	1	0	0	1	0
<i>d</i>	1	0	0	0	1	0
<i>e</i>	0	0	1	1	0	1
<i>f</i>	0	1	0	0	1	0

(a)

<i>a</i>	→	<i>c</i>	→	<i>d</i>	
<i>b</i>	→	<i>c</i>	→	<i>f</i>	
<i>c</i>	→	<i>a</i>	→	<i>b</i>	→ <i>e</i>
<i>d</i>	→	<i>a</i>	→	<i>e</i>	
<i>e</i>	→	<i>c</i>	→	<i>d</i>	→ <i>f</i>
<i>f</i>	→	<i>b</i>	→	<i>e</i>	

(b)

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

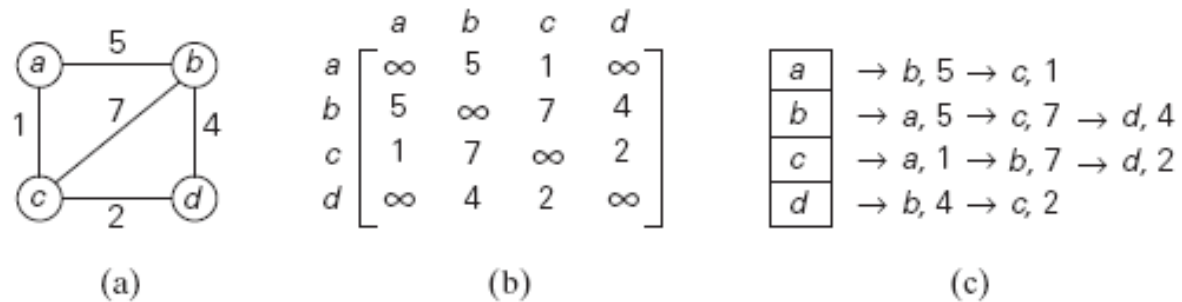
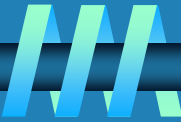


FIGURE 1.8 (a) Weighted graph. (b) Its weight matrix. (c) Its adjacency lists.

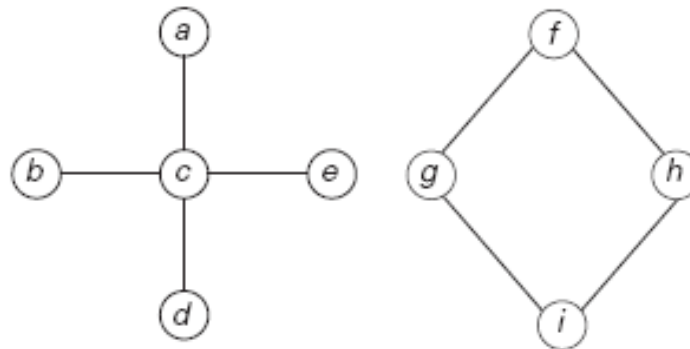
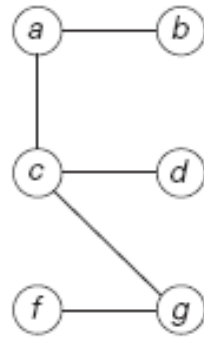
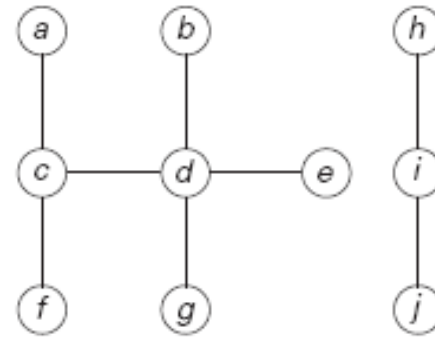


FIGURE 1.9 Graph that is not connected.

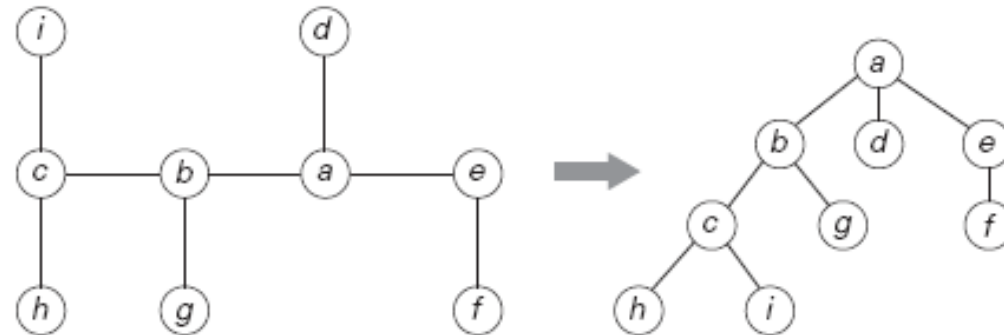


(a)



(b)

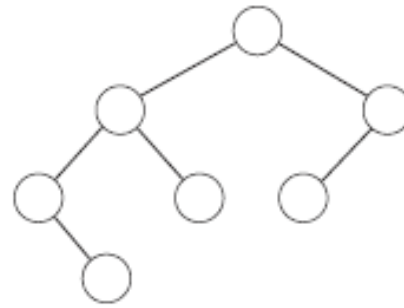
FIGURE 1.10 (a) Tree. (b) Forest.



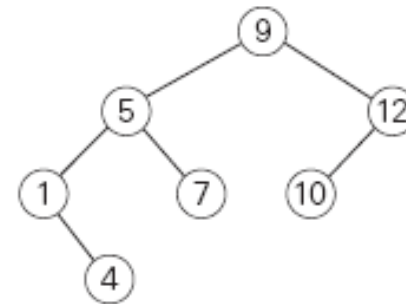
(a)

(b)

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



(a)



(b)

FIGURE 1.12 (a) Binary tree. (b) Binary search tree.

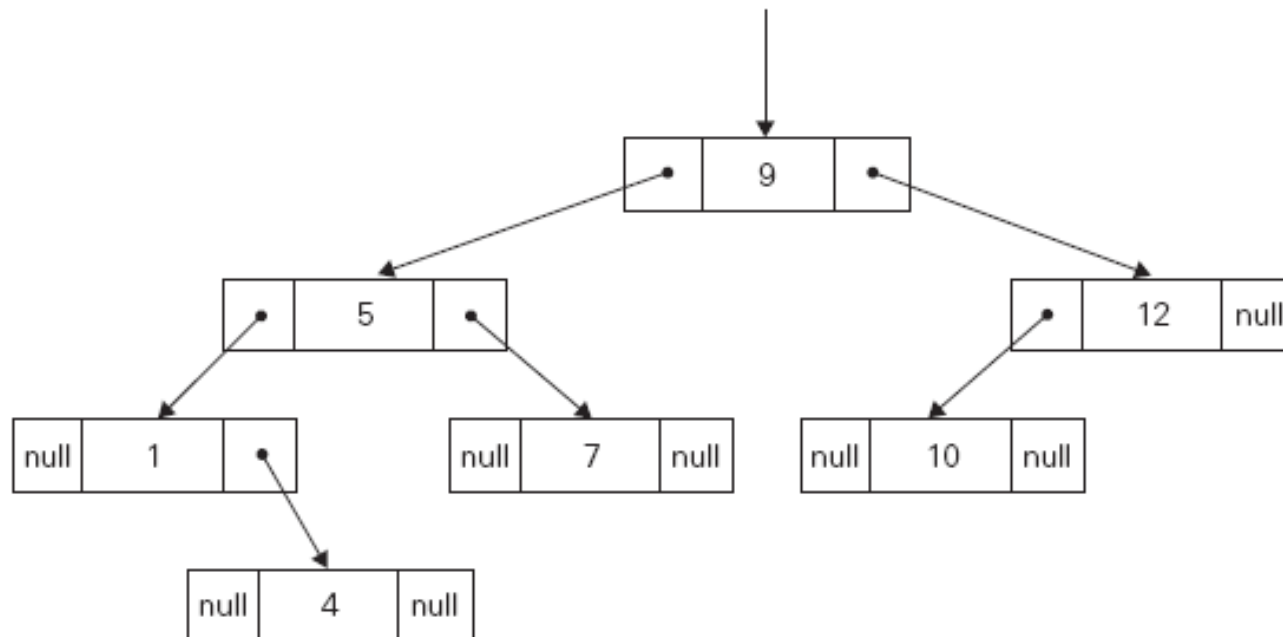


FIGURE 1.13 Standard implementation of the binary search tree in Figure 1.12b.