

INT102

Algorithmic Foundations And Problem Solving

Summary

Dr Jia WANG

Department of Computer Science



西交利物浦大學

Xi'an Jiaotong-Liverpool University



Acknowledgment: The slides are adapted from ones by Dr. Prudence Wong

Desk Assembly Manual

PARTS LIST • Liste des • Lista de piezas

Missing parts? Call 1-800-573-2860
Pièces manquantes? Appeler le 1-800-573-2860
Faltan piezas? Llame al 1-800-573-2860

Desk Parts

- 28 Glass desk top (1)
- 29 Rear frame (1)
- 30 Front frame (1)
- 31 Rear corner braces (2)
- 32 Front corner braces (2)
- 33 Rear center braces (1)
- 34 Screws M6×10 mm (4)
- 35 Screws M6×12 mm (20)
- 36 S-shaped hex wrench (1)

Éléments pour le bureau

- 28 Dessus en verre du bureau (1)
- 29 Structure arrière (1)
- 30 Structure avant (1)
- 31 Entretôises de l'angle arrière (2)
- 32 Entretôises de l'angle avant (2)
- 33 Entretôises arrière centrale (1)
- 34 Vis M6 × 10 mm (4)
- 35 Vis M6 × 12 mm (20)
- 36 Clé hexagonale en S (1)

Partes del escritorio

- 28 Panel superior de vidrio del escritorio (1)
- 29 Bastidor posterior (1)
- 30 Bastidor frontal (1)
- 31 Soportes para los ángulos posteriores (2)
- 32 Soportes para los ángulos frontales (2)
- 33 Soporte posterior central (1)
- 34 Tornillos M6 × 10 mm (4)
- 35 Tornillos M6 × 12 mm (20)
- 36 Llave hexagonal en forma de S (1)

Cabinet Parts

- 1 Glass top (1)
- 2 Front frame (1)
- 3 Upper braces (2)
- 4 Lower braces (2)
- 5 Back frame (1)
- 6 Left side (1)
- 7 Right side (1)
- 8 File drawer left side (1)
- 9 File drawer back (1)
- 10 File drawer right side (1)
- 11 Drawer (1)
- 12 File drawer (1)
- 13 Utility drawer (1)
- 14 Utility drawers (2)
- 15 Utility drawers (2)
- 16 File rail (2)
- 17 Bolts (16)
- 18 Drawer handles (2)
- 19 Front casters (2)
- 20 Back casters (2)
- 21 Flat tipped screws (2)
- 22 Screws M6×15 mm (2)
- 23 Sharp-tipped screws M4×12 mm (12)
- 24 Wrench (1)
- 25 L-shaped hex wrench (1)

Additional Tools required: Phillips screwdriver

Éléments du classeur mobile

- 1 Dessus en verre (1)
- 2 Structure avant (1)
- 3 Entretôises supérieures (2)
- 4 Entretôises inférieures (2)
- 5 Structure arrière (1)
- 6 Côté gauche (1)
- 7 Côté droit (1)
- 8 Côté gauche du tiroir à dossiers (1)
- 9 Arrière du tiroir à dossiers (1)
- 10 Côté droit du tiroir à dossiers (1)
- 11 Fonds du tiroir (1)
- 12 Avant du tiroir à dossiers (1)
- 13 Côté gauche du tiroir de rangement (2)
- 14 Arrière du tiroir de rangement (2)
- 15 Côté droit du tiroir de rangement (2)
- 16 Avant du tiroir de rangement (2)
- 17 Supports dossiers suspendus (2)
- 18 Vis (16)
- 19 Poignée du tiroir (3)
- 20 Roulettes avant (2)
- 21 Roulettes arrière (2)
- 22 Vis à bout plat 5/32 po × 12 mm (6)
- 23 Vis M6 × 12 mm (16)
- 24 Vis M6 × 15 mm (2)
- 25 Vis à bout pointu M4 × 12 mm (12)
- 26 Clé (1)
- 27 Clé hexagonale en L (1)

Outils supplémentaires nécessaires: un tournevis cruciforme

Partes del archivero

- 1 Panel superior de vidrio (1)
- 2 Bastidor frontal (1)
- 3 Soportes superiores (2)
- 4 Soportes inferiores (2)
- 5 Bastidor posterior (1)
- 6 Lado izquierdo (1)
- 7 Lado derecho (1)
- 8 Lado izquierdo del archivero (1)
- 9 Parte posterior del archivero (1)
- 10 Lado derecho del archivero (1)
- 11 Paneles inferiores de las gavetas (3)
- 12 Parte frontal de la gaveta para carpetas colgantes (1)
- 13 Lado izquierdo de las gavetas de uso general (2)
- 14 Panel posterior de las gavetas de uso general (2)
- 15 Panel lateral derecho de las gavetas de uso general (2)
- 16 Panel frontal de las gavetas de uso general (2)
- 17 Riel para las carpetas colgantes (2)
- 18 Pernos (16)
- 19 Agarradera de la gaveta (3)
- 20 Ruedas frontales (2)
- 21 Ruedas posteriores (2)
- 22 Tornillos de punta plana 5/32 pulg. × 12 mm (6)
- 23 Tornillos M6 × 12 mm (16)
- 24 Tornillos M6 × 15 mm (2)
- 25 Tornillos con punta afilada M4×12 mm (12)
- 26 Llave (1)
- 27 Llave hexagonal con forma de L (1)

Herramientas adicionales requeridas: Un destornillador Phillips

File Cabinet Assembly Montage du classeur mobile Ensamblaje del gabinete del archivero

1 Assemble frame Monter la structure Arme el bastidor

Screws M6×12 mm (16)
 Vis M6 × 12 mm (16)
 Tornillos M6 × 12 mm (16)

L shaped wrench (1)
 Clé hexagonale en L (1)
 Llave hexagonal con forma de L (1)

2 Assemble side Fixer les parnis Ensamble los

Sharp-tipped screws M4×12 mm (12)
 Vis à bout pointu M4 × 12 mm (12)
 Tornillos con punta afilada M4 × 12 mm (12)

3 Attach file cabinet casters Fixer les roulettes du classeur mobile Fije las ruedas al archivero

Front casters (2)
 Roulettes avant (2)
 Ruedas frontales (2)

Back casters (2)
 Roulettes arrière (2)
 Ruedas posteriores (2)

Wrench (1)
 Clé (1)
 Llave (1)

ASSEMBLY INSTRUCTIONS
INSTRUCTIONS D'ASSEMBLAGE
INSTRUCCIONES DE ENSAMBLAJE

NT-C3103
48" Steel/Glass Desk
with File Cabinet
Bureau en acier et verre de
48 po avec classeur mobile
Escritorio de acero y vidrio
de 48" con archivero

(Continue on backside)
 (suite au verso)
 (ver al reverso)

What is INT102?

Algorithms + Data Structures + Methods

Complexity Analysis

Algorithms

- Particular problem-solving algorithms
- Sorting/ Searching/Shortest path

Method

- Greedy/Recurrences/Dynamic/Brach-and-bond/Approximation/Backtracking

Data Structures

- Graph

Pseudo Code: conditional

Conditional statement

```
if condition then  
    statement
```

```
if condition then  
    statement  
else  
    statement
```

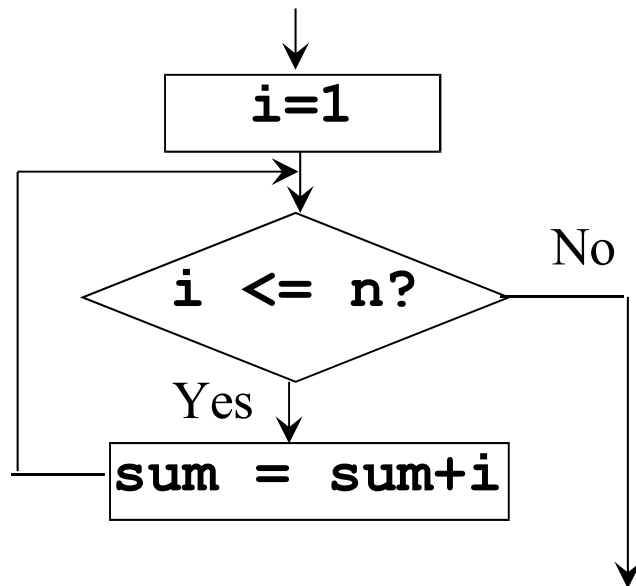
What do these two
algorithms compute?

```
if a < 0 then  
    a = -a  
abs = a  
output abs
```

```
if a > 0 then  
    abs = a  
else  
    abs = -a  
output abs
```

for loop

```
for var = start_value to end_value do  
    statement
```



Computing sum of the first n numbers:

```
input n  
sum = 0  
for i = 1 to n do  
    begin  
        sum = sum + i  
    end  
output sum
```

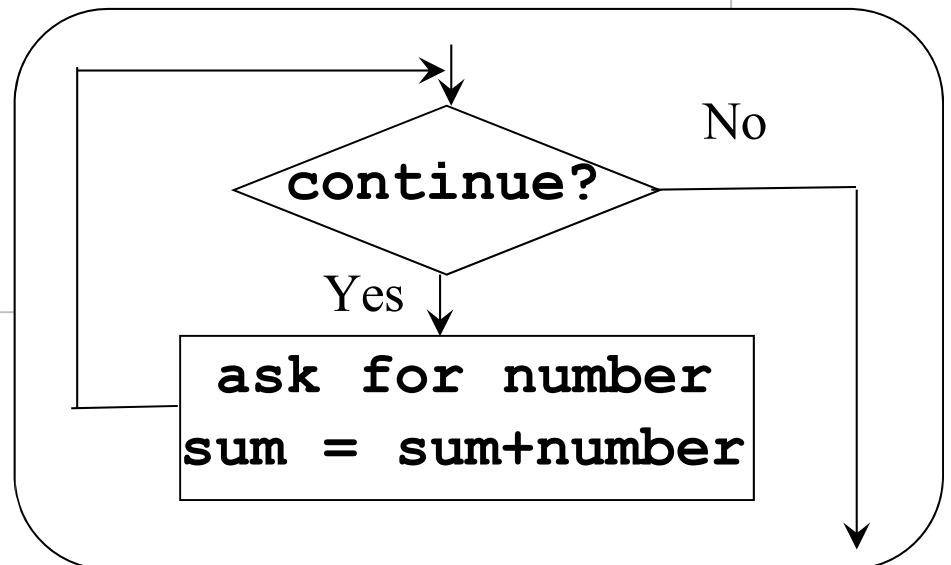
the **loop** is executed for **n** times

while loop

this **loop** is executed for **undetermined** number of times

Computing sum of all (keyboard) input numbers:

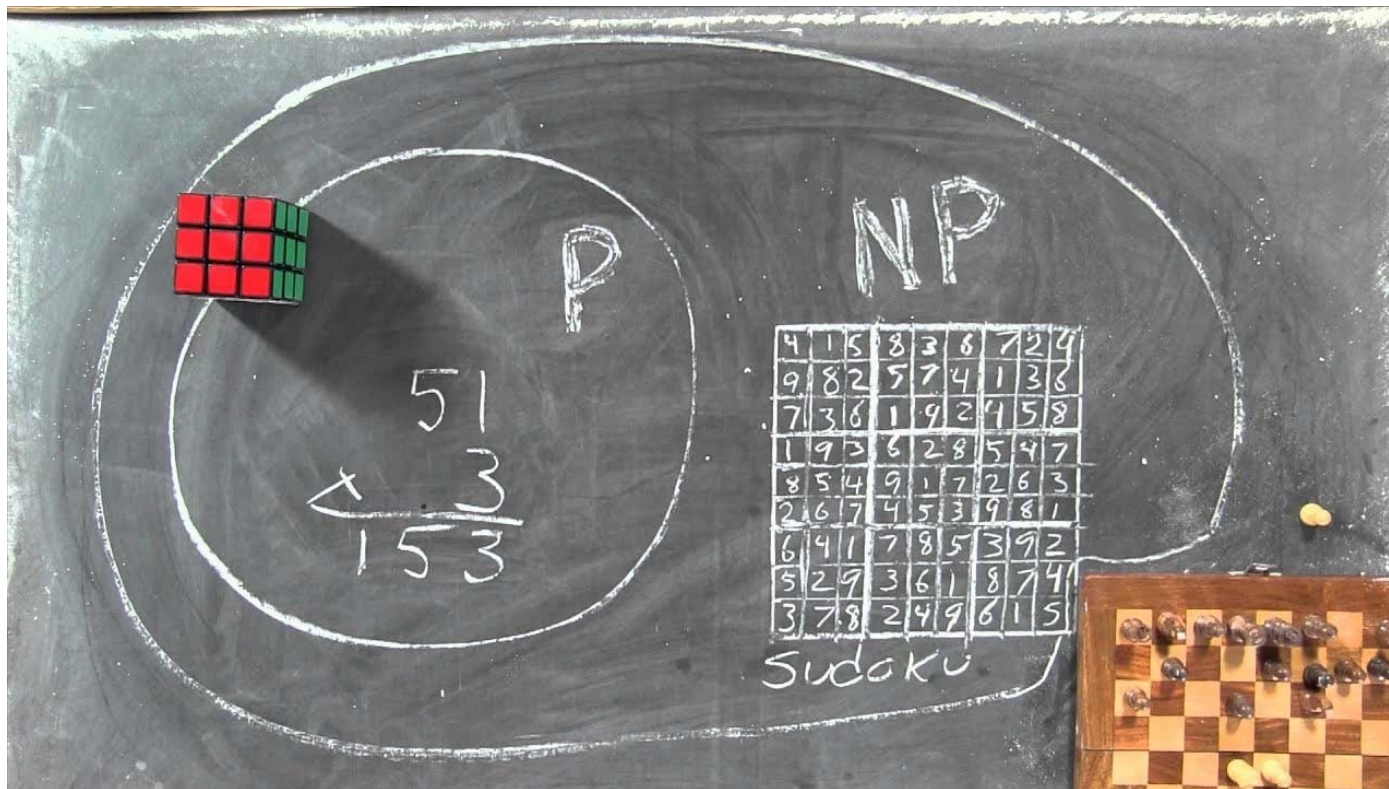
```
sum = 0
while (user wants to continue) do
begin
    ask for a number
    sum = sum + number
end
output sum
```



Time complexity

- Big O notation ...**

Able to carry out simple asymptotic analysis of algorithms



Particular problem-solving algorithms

Searching

- ⑩ **Input:** a sequence of n numbers a_0, a_1, \dots, a_i and a number X
- ⑩ **Output:** determine whether X is in the sequence or not
- ⑩ **Algorithm (Linear search):**
 1. Binary Search
 2. Linear Search
 3. Pattern Searching

Sorting

- ⑩ **Input:** a sequence of n numbers a_0, a_1, \dots, a_{n-1}
- ⑩ **Output:** arrange the n numbers into ascending order, i.e., from smallest to largest
- ⑩ **Example:** If the input contains 5 numbers ~~12~~56, 43, 200, 10, then the output should be 10, 43, 56, 132, 200

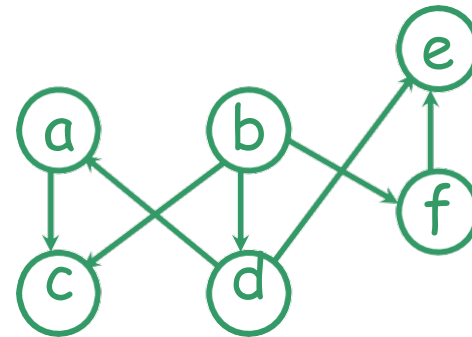
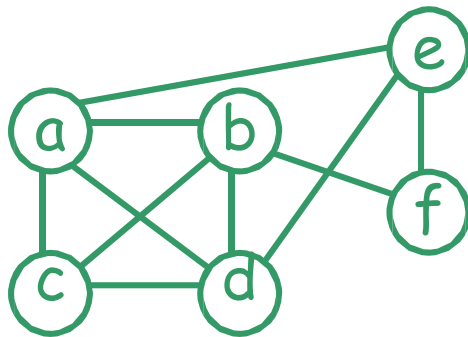
There are many sorting algorithms:
insertion sort, selection sort, bubble sort,
merge sort, quick sort

Graph ... DATA STRUCTURE

Graphs

Graph theory - an old subject with many modern applications.

An *undirected* graph $G=(V,E)$ consists of a set of vertices V and a set of edges E . Each edge is an *unordered* pair of vertices. (E.g., $\{b,c\}$ & $\{c,b\}$ refer to the same edge.)



A *directed* graph $G=(V,E)$ consists of ... Each edge is an *ordered* pair of vertices. (E.g., (b,c) refer to an edge from b to c.)

Learning outcomes

- Able to tell what is an undirected graph and what is a directed graph
 - Know how to represent a graph using matrix and list
- Understand what Euler path / circuit and able to determine whether such path / circuit exists in an undirected graph
- Able to apply BFS and DFS to traverse a graph
- Able to tell what a tree is

Methods

1. Greedy methods

- Understand what greedy method is
- Able to apply Prim's algorithm to find minimum spanning tree
- Able to apply Kruskal's algorithm to find minimum spanning tree
- Able to apply Dijkstra's algorithm to find single-source shortest-paths

2. Dynamic Programming

- Understand the basic idea of dynamic programming
- Able to apply dynamic programming to solve problems
 - Assembly line/ Fibonacci numbers/Global alignment/Local alignment/LCS/Knapsack problem

3. Backtracking

- Construct the state-space tree
 - nodes: partial solutions
 - edges: choices in extending partial solutions
- Explore the state space tree using depth-first search
- "Prune" nonpromising nodes
 - dfs stops exploring subtrees rooted at nodes that cannot lead to a solution and backtracks to such a node's parent to continue the search

4. Branch-and-Bound

- ❑ An enhancement of backtracking
- ❑ Applicable to optimization problems
- ❑ For each node (partial solution) of a state-space tree, computes a bound on the value of the objective function for all descendants of the node (extensions of the partial solution)
- ❑ Uses the bound for:
 - ruling out certain nodes as “nonpromising” to prune the tree - if a node's bound is not better than the best solution seen so far
 - guiding the search through state-space

5. Approximation

- ⑩ Find a “good” solution fast
 - sufficient for many applications
 - we often have inaccurate data to start with, so approximation may be as good as optimal solution

INT102

END OF TEACHING

What we have learnt

Methodology problems	Asymptotic idea	Brute force	Divide & Conquer	Dynamic Programming	Greedy	Space/Time	Branch & Bound	Backtracking	Complexity Theory
Efficiency	Big-O								
Sorting		Selection/ Bubble/insertion	Merge-sort			Count sorting			
Searching			Binary-searching						
String		searching		Alignment/LCS		Horspool algorithm			
Graph/Combinatory		DFS/BFS		Floyd's Algorithm/ Assembly-line Knapsack	MST(Prim's/ Kruskal's) Dijkstra's For Shortest path		Traveling salesman, Job assignment	n-Queens Sum of subset Hamiltonian Problem	Approximation: TSP problem: Nearest-Neighbor/twice round/fragment algorithm
Complexity									P/NP Circuit-SAT/3-SAT

How will it be assessed

- Two assignments (20% of the final mark)
 1. Assignment 1 (week 6 - week 7) (10%)
 2. Assignment 2 (week 11 - week 12) (10%)
- Final Examination (80% of the final mark)

written examination: 70% MCQ's + 30% Problem Solving

This is an **CLOSED BOOK** examination.

Questions 1 to 4 refer to the following algorithm.

Algorithm: $P(a[l \dots r])$

Input: an array $a[l \dots r]$ of real numbers

begin

if $l = r$ **return** l

else

$ll = P(a[l \dots \lfloor (r+l)/2 \rfloor])$

$rr = P(a[\lfloor (r+l)/2 \rfloor + 1 \dots r])$

if $a[ll] > a[rr]$ **return** ll

else return rr

end

<p>1. Which algorithm design technique is employed in the above algorithm?</p> <p>[A] Brute Force technique</p> <p>[B] Greedy technique</p> <p>[C] Divide- and-Conquer</p> <p>[D] Dynamic Programming</p> <p>[E] Time and Space trade-off</p>	2.5
<p>2. For $n = 2^k$ and $k \geq 1$, the time complexity of the algorithm can be best expressed by</p> <p>[A] $T(n) = T(n/2) + 1$</p> <p>[B] $T(n) = T(n/2)$</p> <p>[C] $T(n) = 2T(n/2) + 1$</p> <p>[D] $T(n) = 2T(n/2)$</p> <p>[E] None of the above</p>	

Analyzing the Time Complexity

To determine the time complexity, let's denote $T(n)$ as the time complexity of the algorithm for an array of size n .

1. **Base Case:** When $l = r$ (i.e., the subarray has only one element), the algorithm takes constant time:

$$T(1) = O(1)$$

2. **Recursive Case:** When $l \neq r$, the algorithm divides the array into two subarrays of approximately equal size (each of size $n/2$), solves the problem for each subarray, and then combines the results by comparing two elements:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$$

The $O(1)$ term represents the time taken to compare the two elements and return the index of the larger one.

Solving the Recurrence Relation

The recurrence relation for $T(n)$ is:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$$

3. The time complexity of the algorithm is	2.5
[A] $O(n)$ [B] $O(\log n)$ [C] $O(n^2)$ [D] $O(n \log n)$ [E] None of the above	
4. What is the output of the algorithm for the input $a[0..7] = [12, 12, 12, 12, 12, 12, 12, 12]$?	2.5
[A] 1 [B] 3 [C] 5 [D] 7 [E] 12	

Algorithm Explanation

The algorithm recursively finds the index of the maximum element in the array by dividing the array into smaller subarrays until each subarray has only one element. It then compares the maximum elements of these subarrays and returns the index of the larger element. If the elements are equal, it returns the right index.

item	weight	value
1	2	\$12
2	1	\$10
3	3	\$20

The Knapsack Capacity $W=3$

Let $V[i, j]$ be the value of the most valuable subset of the first i items that fit into the Knapsack of capacity j . Then $V[i, j]$ can be recursively defined as follows:

$$V[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max \{V[i-1, j], v_i + V[i-1, j-w_i]\} & \text{if } j - w_i \geq 0 \\ V[i-1, j] & \text{if } j - w_i < 0 \end{cases}$$

For the above instance, the following is an incomplete table for $V[i, j]$ ($i=0, 1, 2, 3$; $j=0, 1, 2, 3$)

		capacity j			
Item	i	0	1	2	3
	0	0	0	0	
$w_1=2, v_1=12$	1	0	0	12	12
$w_2=1, v_2=10$	2	0	10		22
$w_3=3, v_3=20$	3	0	10	12	

22. What is the value of $V[1, 2]$?	2.5
<div>[A] 12</div> <div>[B] 10</div> <div>[C] 22</div> <div>[D] 0</div> <div>[E] 24</div>	
23. What is the value of the most valuable subset?	2.5
<div>[A] 12</div> <div>[B] 10</div> <div>[C] 22</div> <div>[D] 0</div> <div>[E] 24</div>	

Algorithm: LIS($A[0\dots n-1]$)

// Input: An array $A[0\dots n-1]$ of n numbers

Begin

$\text{length}[0\dots n-1]$ = array of size n filled with 1

 // $\text{length}[i]$ will hold the length of the LIS ending at position i

 for i from 1 to $n-1$ do

 for j from 0 to $i-1$ do

 if $A[i] > A[j]$ and $\text{length}[i] < \text{length}[j] + 1$ then

$\text{length}[i] = \text{length}[j] + 1$

$\text{maxLength} = 0$

 for i from 0 to $n-1$ do

 if $\text{maxLength} < \text{length}[i]$ then

$\text{maxLength} = \text{length}[i]$

 return maxLength

End

Which algorithm design technique is employed in the above algorithm?

- [A] Brute Force technique
- [B] Greedy technique
- [C] Divide- and-Conquer
- [D] Dynamic Programming
- [E] Ad hoc technique

The output of the algorithm is

- [A] The sum of the longest increasing subsequence.
- [B] The length of the shortest increasing subsequence.
- [C] The length of the longest increasing subsequence.
- [D] The number of increasing subsequences.
- [E] None of above.