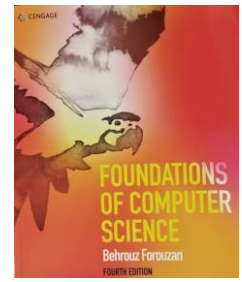
Chapter 8



Algorithms
演算法

Objectives 學習目標

After studying this chapter, students should be able to:

- **Define an algorithmand relate it to problem solving.定義演算法**
- **Define three construct and describe their use in algorithms.三種結構使用於演算法**
- **Describe UML diagrams and pseudocode and how they are used in algorithms.演算法表示：流程圖與虛擬碼**
- **List basic algorithms and their applications.**
- **Describe the concept of sorting and understand the mechanisms behind three primitive sorting algorithms.排序**
- **Describe the concept of searching and understand the mechanisms behind two common searching algorithms.搜尋**
- **Define subalgorithms and their relations to algorithms.副程式**
- **Distinguish between iterative循環 and recursive遞迴 algorithms**

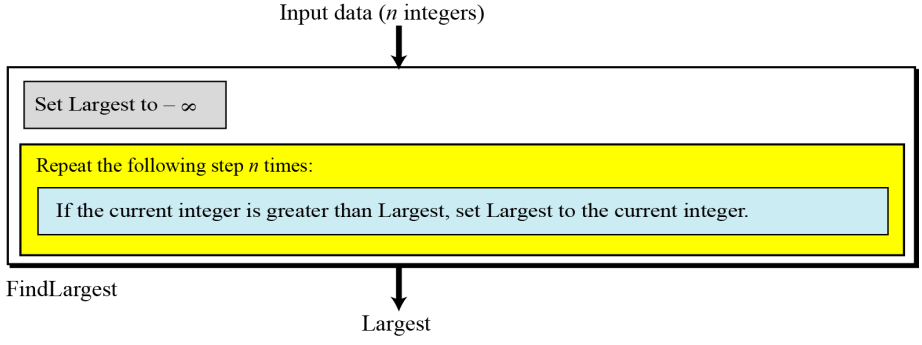
Generalization 一般化

Is it possible to **generalize the algorithm**? We want to find the largest of *n* positive integers, where *n* can be 1000, 1,000,000, or more. Of course, we can follow the above method and **repeat each step**. But if we change the algorithm to a program, then we need to actually type the actions for *n* steps!

There is a better way to do this. We can **tell the computer to initial the state and repeat the steps *n* times** 初始狀態及重複**n**次**.** We now include this feature in our pictorial algorithm.

8.3

Generalization of FindLargest




```
FindLargest (List[n])    // Given: List[0] ~ List[n-1]
{    Largest = -99999; 初始狀態
     for (i = 0 to n-1)   重複n次
     {   if (List [i] > Largest)  Largest = List[i];
     }
     return Largest;
}
```

8.4

演算法正式定義

Now that we have discussed the concept of **an algorithm** and shown its representation, here is a more **formal definition**:

An algorithm is an ordered set of unambiguous steps that produces a result and terminates in a finite time. 一演算法由一組有次序而非模稜兩可的步驟組成，能在有限時間內完成，並產生正確的結果

8.5

Sorting Algorithms 排序演算法

One of the most common applications in computer science is **sorting, which is the process by which data is arranged according to its values** 將資料依照其值而排列. People are surrounded by data. If the data was not ordered, it would take hours and hours to find a single piece of information.如資料為排序，尋找某一資料很困難. Imagine the difficulty of finding someone's telephone number in a telephone book that is not ordered.

In this section, we introduce **three basic sorting algorithms** 三個基本排序方法: **selection sort選擇排序法**, **bubble sort氣泡排序法** and **insertion sort插入排序法**. These three sorting algorithms are the foundation of faster sorting algorithms used in computer science today.

8.6

The Pseudocodes of Selection Sort 選擇排序法虛擬碼

```
SelectionSort (List[n]) //    Given: List[0] ~ List[n-1]
{    for (i = 0 to n-1)
     {    smallest = List [i];     k_min = i;
          for (j = i+1 to n-1)
          {       if (List[j] < smallest)
                  {       smallest = List[j];    k_min = j;
                  }
          }
          temp = List[i];
          List[i] = List[k_min] ;  List[k_min] = temp;
     }
}
```

8.7

Searching Algorithms 搜尋演算法

Another common algorithm in computer science is searching, which is the process of finding the location of a target among a list of objects. In the case of a list, searching means that **given a value**, we want to find the location of the first element in the list that contains that value. 搜尋乃找 一筆資料位在哪裡

There are **two basic searches for lists兩種基本搜尋方法: sequential search 順序搜尋 and binary search二元搜尋**. Sequential search can be used to locate an item in any list, whereas binary search requires the list first to be sorted.

8.8

Binary Search for a given sorted 1D-array A[19]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
5	7	10	19	27	33	38	42	47	54	62	65	69	71	77	81	85	91	97

```
Sequential_Serach (A[n],91)
{   count = 0;
    for (i=0 to n-1)
    {   count = count + 1;
        if (A[i]==91)  return count;
     }
}
```

```
Binary_Serach (A[n],91)
{   low=0; high=n-1; count=0;
    while (low <= high)
    {   mid = (low+high) / 2; count = count+1;
        if (A[i]==91)  return count;
        else  if (A[i]<=91)  low = mid+1;
              else           high = mid-1;
     }
}
```

8.9

Recursive Solution to Factorial Problem 階乘問題的遞迴演算法

The recursive solution does not need a loop, as the recursion concept itself involves repetition. 階乘問題的遞迴演算法: 只需重複呼叫本身而不需要迴圈。

Algorithm 8.7 Pseudocode for recursive solution of factorial problem

```
Algorithm: Factorial (n)
Purpose: Find the factorial of a number using recursion
Pre: Given: n
Post: None
Return: n!
{
      if (n = 0)        return 1
      else              return n × Factorial (n –1)
}
```

```
Factorial (n)
{    if (n == 0)  return (1);
     else         return (n * Factorial (n-1));
}
```

8.10

Review Questions

- What is the **formal definition** of **algorithm**?
- Please show **two kinds of algorithm representation**.
- Please list the detailed steps of **selection sort** for the numbers, 56, 78, 25, 39, 99, 67, and 15.
- Please list the detailed steps of **bubble sort** for the numbers, 56, 78, 25, 39, 99, 67, and 15.
- Please list the detailed steps of **insertion sort** for the numbers, 56, 78, 25, 39, 99, 67, and 15.
- Please give **the algorithm** of **binary search**.
- Please give the **iterative algorithm** of ***n* factorial**.
- Please give the **recursive algorithm** of ***n* factorial**.