北一女中 2014
資訊選手培訓營
0818~0822

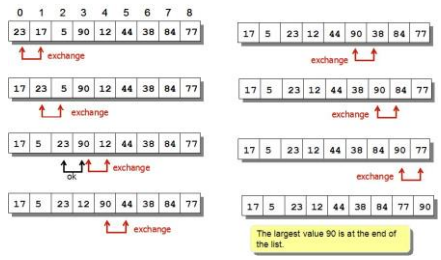
排序演算法

Sorting Algorithm Animations

http://www.sorting-algorithms.com/

排序的方法

- 泡沫排序法(bubble sort)
- 選擇排序法(selection sort)
- 插入式排序法(insertion sort)
- 謝爾排序法(shell sort)
- 快速排序法(quick sort)
- 合併排序法(merge sort)
- 累堆排序法(heap sort)
- 基數排序法(radix sort)



泡沫排序法(BUBBLE SORT)

泡沫排序法(bubble sort)

- 假設現在我們需要將 n 筆資料 A_1、A_2、.....、A_n 由大排到小。
  - 在資料陣列中，由左向右比較，只要資料值比右邊的小，二者就交換
  - 在第一輪比完，最右邊的是最小值
  - 第二輪比較時比到倒數第二筆結束，出現第二小的數值
  - 同樣的，在n-1次迴圈後，資料排序完成
  - 時間複雜度：$O(n^2)$

氣泡排序法(Bubble Sort)
練習題

1 43 6 79 50 2

氣泡排序法(Bubble Sort)-
第一輪的比較與交換

- 1 43 6 79 50 2

氣泡排序法(Bubble Sort)-
第二輪的比較與交換

- 43 6 79 50 2 1

氣泡排序法(Bubble Sort)-
Worse Case

```
Function bubbleSort(Type data[1..n])
    Index i, j;                                  // 1
    For i from n to 2 do                         // n - 1
        For j from 1 to i - 1 do                 // n(n - 1) / 2
            If data[j] > data[j + 1] then        // n(n - 1) / 2
                Exchange data[j] and data[j + 1] // n(n - 1) / 2
End
```

- 執行次數總和為：
  $W(n) = 1 + (n - 1) + n(n - 1) / 2 + n(n - 1) / 2 + n(n - 1) / 2 = (3n^2 - n) / 2 \in O(n^2)$
- 平方時間（quadratic time）的演算法

氣泡排序法(Bubble Sort)-
Best Case

```
Function bubbleSort(Type data[1..n])
    Index i, j;                                  // 1
    For i from n to 2 do                         // n - 1
        For j from 1 to i - 1 do                 // n(n - 1) / 2
            If data[j] > data[j + 1] then        // n(n - 1) / 2
                Exchange data[j] and data[j + 1] // 0
End
```

- 執行次數總和為
  $B(n) = 1 + (n - 1) + n(n - 1) / 2 + n(n - 1) / 2 + 0 = n^2 \in O(n^2)$
- 複雜度依舊為平方時間

Selection Sort

```
SelectionSort(int A[], int N)
{
 for (int Last = N-1; Last >=1;
      --Last)
 {
  int L = Largest (A, Last+1);
  Swap (A[L], A[Last]);
 }
}
```

10	20	31	5	12
10	20	12	5	31
10	5	12	20	31
10	5	12	20	31
5	10	12	20	31

6

選擇排序法(SELECTION SORT)

選擇排序法(Selection Sort)

- 原理：
  - 將資料分為「已排序」與「未排序」兩部份
  - 從「未排序」的資料中找出最大（最小）值，放入「已排序」資料的最後端。
  - 如是進行，直到排序結束（未排序資料為空）為止。

選擇排序法(Selection Sort)

- 資料由大排到小

19 58 33 41 28 14 53 84

選擇排序法(Selection Sort)
運作過程

選擇排序法(Selection Sort)

```
Function selectionSort(Type data[1..n])
    Index i, j, max
    For i from 1 to n do
        max = i
        For j from i + 1 to n do
            If data[j] > data[max] then
                max = j

        Exchange data[i] and data[max]
End
```

選擇排序法(Selection Sort)
最差情況

- 所有資料的順序剛好完全相反
- 使用選擇排序法將 n 筆資料排序

```
Function selectionSort(Type data[1..n])
    Index i, j, max                          // 1
    For i from 1 to n do                     // n
        max = i                              // n
        For j from i + 1 to n do             // n(n - 1) / 2
            If data[j] > data[max] then      // n(n - 1) / 2
                max = j                      // n(n - 1) / 2

        Exchange data[i] and data[max]       // n
End
```

- 執行次數總和
  $W(n) = 1 + n + n + n(n - 1) / 2 + n(n - 1) / 2 + n(n - 1) / 2 + n = (3n^2 + 3n - 2) / 2 \in O(n^2)$。

插入式排序法(INSERTION SORT)

插入式排序法(insertion sort)

已排序　插入的位置　目前掃瞄的位置　run

排序前：11,6,59,65,6,94,93,63,69,95

Step0:
11 6 59 65 6 94 93 63 69 95

Step1: 處理值 = 6
6 11 59 65 6 94 93 63 69 95

Step2: 處理值 = 59
6 11 59 65 6 94 93 63 69 95

Step3: 處理值 = 65
6 11 59 65 6 94 93 63 69 95

Step4: 處理值 = 6
6 6 11 59 65 94 93 63 69 95

Step5: 處理值 = 94
6 6 11 59 65 94 93 63 69 95

http://notepad.yehyeh.net/Content/Algorithm/Sort/Insertion/1.php

插入式排序法(insertion sort)

- 原理：
  - 將資料分為「已排序」與「未排序」兩個部份。
  - 再將未排序資料中的第一筆資料插入到已排序資料的適當位置。

插入式排序法(insertion sort)

1. 將資料分成**已排序**、**未排序**兩部份
2. 依序由**未排序**中的**第一筆**(正處理的值)，插入到**已排序**中的適當位置
   - 插入時**由右而左**比較，直到遇到第一個比**正處理的值**小的值，再插入
   - 比較時，若遇到的值比**正處理的值**大或相等，則將值往右移

插入式排序法(insertion sort)

```
Function insertionSort(Type data[1..n])
    Index i, j;
    Type value;
    For i from 2 to n do
        value = data[i];

        j = i - 1;
        While j >= 1 and data[j] > value do
            data[j + 1] = data[j];
            j = j - 1;

        data[j + 1] = value;
End
```

插入式排序法(insertion sort)
練習題

35 42 18 91 65 70 52 55 40 82

	0	1	2	3	4	5	6	7	8	9	說明
i	35	42	18	91	65	70	52	55	40	82	
1	35	**42**									42 沒有任何移動
2	**18**	35	42								18 移到第0個位置
3	18	35	42	**91**							91 維持不變
4	18	35	42	**65**	91						65 移到第三個位置
5	18	35	42	65	**70**	91					70 移到第四個位置
6	18	35	42	**52**	65	70	91				52 移到第三個位置
7	18	35	42	52	**55**	65	70	91			55 移到第四個位置
8	18	35	**40**	42	52	55	65	70	91		40 移到第二個位置
9	18	35	40	42	52	55	65	70	**82**	91	82 移到第八個位置

圖9.2 插入式排序

插入式排序法(insertion sort)
Best Case

- 資料都已排序的最佳情況下，資料大小為 n

```
Function insertionSort(Type data[1..n])
    Index i, j;                                  // 1
    Type value;                                  // 1
    For i from 2 to n do                         // n - 1
        value = data[i];                         // n - 1

        j = i - 1;                               // n - 1
        While j >= 1 and data[j] > value do      // n - 1
            data[j + 1] = data[j];               // 0
            j = j - 1;                           // 0

        data[j + 1] = value;                     // n - 1
End
```

- 執行次數總和：B(n) = 1 + 1 + (n - 1) + (n - 1) + (n - 1) + (n - 1) + (n - 1) = 5n - 3 ∈ O(n)
- 複雜度為線性時間

插入式排序法(insertion sort) Worse Case

- 資料完全反序的最差情況

```
Function insertionSort(Type data[1..n])
    Index i, j;                                 // 1
    Type value;                                 // 1
    For i from 2 to n do                        // n - 1
        value = data[i];                        // n - 1

        j = i - 1;                              // n - 1
        While j >= 1 and data[j] > value do     // n(n - 1) / 2
            data[j + 1] = data[j];              // n(n - 1) / 2
            j = j - 1;                          // n(n - 1) / 2

        data[j + 1] = value;                    // n - 1
End
```

- 執行次數總和：W(n) = 1 + 1 + (n - 1) + (n - 1) + (n - 1) + n(n - 1) / 2 + n(n - 1) / 2 + n(n - 1) / 2 + (n - 1) = $(3n^2 + 3n - 4) / 2 \in O(n^2)$
- 複雜度為平方時間

插入式排序法(insertion sort) 時間複雜度

- Best Case：O(n)
  - 當資料的順序恰好為由小到大時，每回合只需比較1次
- Worst Case：$O(n^2)$
  - 當資料的順序恰好為由大到小時，第i回合需比i次
- Average Case：$O(n^2)$
  - 第n筆資料，平均比較n/2次

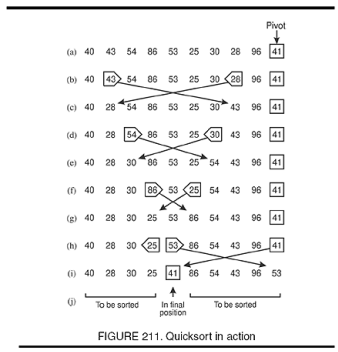


FIGURE 211. Quicksort in action

快速排序法(QUICK SORT)

快速排序法(quick sort)

- 排序演算法中，平均效能最佳者
- 基本方法：採用遞迴的觀念
  - 選取第一個數值為基準(pivot)
  - 將數值資料切割成兩部分：
    - 第一部份的所有元素值都小於基準值
    - 第二部份的所有元素值都大於基準值
  - 兩個部分再分別執行quick sort

快速排序法(quick sort)

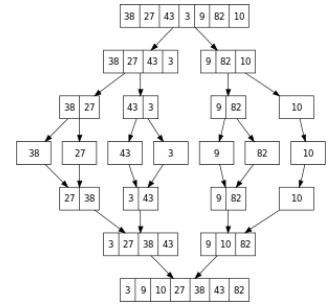
http://notepad.yehyeh.net/Content/Algorithm/Sort/Quick/Quick.php

快速排序法(quick sort) 時間複雜度

- 最佳： O(n log n)
  每次切割成平均兩半
- 平均：O(n log n)
- 最差：$O(n^2)$
  每次切割成1個與n-1個

快速排序法(quick sort)
競賽試題

13. 假設陣列的元素 A[1]=9, A[2]=6, A[3]=10, A[4]=8, A[5]=16, A[6]=7, A[7]=23, A[8]=20, A[9]=15, A[10]=3，則當利用快速排序法(Quick Sort)將此陣列的元素A[1]到 A[10]由小到大排序時，若當"9"第一次被交換到 A[5]這個位置後，排序的過程立即因故而終止，則此時陣列內的元素 A[1]的值為___(19)___，又A[6]的值為___(20)___。



合併排序法(MERGE SORT)

合併排序法(merge sort)

- 將數列對分成**左子數列**、**右子數列**
- 分別對**左子數列**、**右子數列**作上一個步驟 ⇒ 遞迴(Recursive)
  - 直到**左子數列**、**右子數列**被分割成只剩一個元素為止
  - 將僅剩的一個元素作為遞迴的結果回傳
- 對回傳的**左子數列**、**右子數列依大小排列合併**
- 將**合併**的結果作為遞迴的結果回傳

合併排序法(merge sort)

- 將**左子數列**及**右子數列**依大小合併成一個新的數列 若**左子數列**的數值都已填到新的數列 ⇒ 將**右子數列**中未填過的最小值填入新數列
- 若**右子數列**的數值都已填到新的數列 ⇒ 將**左子數列**中未填過的最小值填入新數列
- 將**左子數列**及**右子數列**中，未填過的最小值填到新的數列

合併排序法(merge sort)

Demo:

leftData rightData
排序前：37,37,95,54
Step1: 37 37 95 54
Call mergeSort(37,37) Step1-1
Call mergeSort(95,54) Step1-2
Step1-1: 37 37
Call mergeSort(37) Step1-1-1
Call mergeSort(37) Step1-1-2
Step1-1-1: 37
Step1-1-2: 37
Step1-1 Merge: [37] [37]
Return Merge Result to Step1: 37 37

http://notepad.yehyeh.net/Content/Algorithm/Sort/Merge/Merge.php

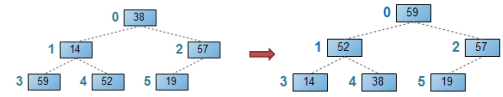
合併排序法(merge sort)
時間複雜度(Time Complexity)

- Best Case：O(n log n)
- Worst Case：O(n log n)
- Average Case：O(n log n)

T(n)
= MergeSort(左子數列) + MergeSort(右子數列) + Merge
= T(n/2) + T(n/2) + c×n
= **O(n $\log_2$n)**

合併排序法(merge sort)

- quick sort的缺點：
  - 最差狀況是O(n^2)
  - 也就是無法避免切割陣列是否平均
- 採用合併排序法
  - 簡化切割原則，每次都切割成一半
  - 以遞迴方式處理
  - 時間複雜度：O(n log n)



堆積排序法(HEAP SORT)

堆積樹(Heap Tree)

- 堆積樹(Heap Tree)： 二元樹的一種 ⇒ 每個父節點最多兩個子節點
- 堆積樹為**完全二元樹(Complete Binary Tree)**的一種
- **最小堆積(Min Heap)**：父節點的值小於子節點
  - **樹根(root)**一定最所有節點的**最小值**
- **最大堆積(Max Heap)**：父節點的值大於子節點
  - **樹根(root)**一定最所有節點的**最大值**
- 兄弟節點的大小不重要
- 堆積排序法為選擇排序法的改良

二元樹調整為Max Heap

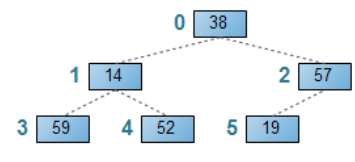
- 二元樹有 floor(n/2)個**內部節點**
- **由後往前**以每個內部節點為Root，作**堆積化(Heapify)**

堆積化(Heapify)

- 令Root的左、右子樹皆符合Heap，僅Root不符合Heap
- 令Root、左子元素、右子元素3個元素中，最大者為MaxNode
- 若Root = MaxNode ⇒ 結束
- 若左子元素或右子元素 = MaxNode
  - 將Root與MaxNode作對調(Swap)
  - 若對調完的Root有子節點 ⇒ 對原來的Root作Heapify

堆積化(Heapify) 範例1

- 二元樹調整為Max Heap

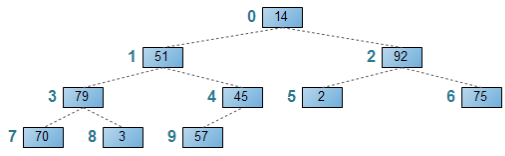


原始二元樹

http://notepad.yehyeh.net/Content/Algorithm/Sort/Heap/Heap.php

堆積化(Heapify) 範例2

- 二元樹調整為Max Heap

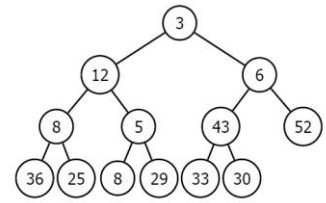


原始二元樹

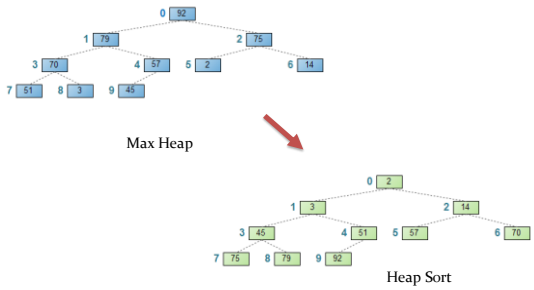
http://notepad.yehyeh.net/Content/Algorithm/Sort/Heap/Heap.php

堆積化(Heapify) 競賽試題

3. 若有一棵二元樹如下圖，經最小的變動而調整成為一個最大堆積樹(max heap tree，意即子節點永遠不大於父節點的堆積樹)後，則最頂點的節點的值為 _____。



Heap Sort



http://notepad.yehyeh.net/Content/Algorithm/Sort/Heap/Heap.php

堆積排序法 時間複雜度(Time Complexity)

- Best Case：O(n log n)
- Worst Case：O(n log n)
- Average Case：O(n log n)
- 說明：
  - 建立MaxHeap： O(n)
  - 執行n-1次Delete Max：(n-1) × O(log n) = O(n log n)
  - O(n) + O(n log n) = O(n log n)

各種排序法比較

- 平均時間複雜度由高到低為：
- 氣泡排序 $O(n^2)$
- 選擇排序 $O(n^2)$
- 插入排序 $O(n^2)$
- 合併排序 $O(n \log n)$
- 堆排序 $O(n \log n)$
- 快速排序 $O(n \log n)$

【C++】使用STL內建的sort()、reverse()函式做到排序功能

- #include <algorithm> 這個標頭檔
- sort() – 由小至大排序
- reverse() – 由大至小排序
- sort(array_begin,array_end);
- reverse(array_begin,array_end);
- array_begin的部份就是開始排序的地方，array_end就是排序的結尾，

排序演算法 培訓練習

請寫下各種演算法的排序過程：

Bubble sort

1 43 6 79 50 2

選擇排序法(Selection Sort)

19 58 33 41 28 14 53 84

插入式排序法(insertion sort)

35 42 18 91 65 70 52 55 40 82

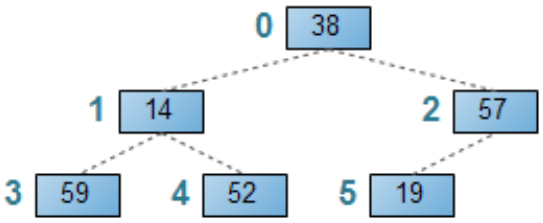
快速排序法(quick sort)

45 68 37 71 76 9 27 27 73 7

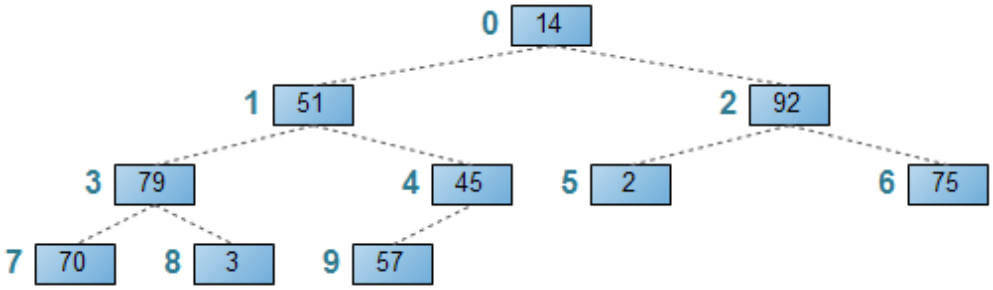
合併排序法(merge sort)

38 27 43 3 9 82 10

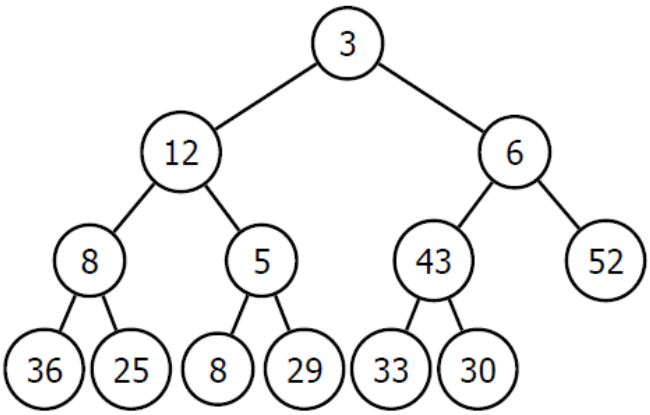
二元樹調整為 Max Heap



二元樹調整為 Max Heap



3. 若有一棵二元樹如下圖，經最小的變動而調整成為一個最大堆積樹(max heap tree，意即子節點永遠不大於父節點的堆積樹)後，則最頂點的節點的值為 _____。



【C++】使用 STL 內建的 sort()、reverse()函式做到排序功能

```
#include <cstdlib>
#include <iostream>
#include <algorithm>            //務必引用 algorithm

using namespace std;

int main()
{
	int x[5]={15,20,5,1,65};
	int x_len=5;

	cout<<"排序前：x=";
	for(int i=0;i<x_len;i++){
		cout<<x[i]<<",";
	}
	cout<<endl;

	cout<<"升冪排序後：x=";
	sort(x,x+x_len);            //排序 x 陣列
	for(int i=0;i<x_len;i++){
		cout<<x[i]<<",";
	}
	cout<<endl;

	cout<<"降冪排序後：x=";
	reverse(x,x+x_len);         //把 x 陣列內容反轉
	for(int i=0;i<x_len;i++){
		cout<<x[i]<<",";
	}
	cout<<endl;

	system("PAUSE");
	return 0;
}
```

資料結構與演算法複習試題(出自：全國資訊競賽 89, 91、IOI 2002, 2003)

Sort, Search && Algorithm

1. 某校有學生20000 人，且已按姓氏筆劃排序, 以二分搜尋法找學生姓名及資料, 最多需要比較幾次?
   (a) 8
   (b) 15
   (c) 10000
   (d) 20000

2. 從n個數中找出第二小的數最少可在幾次比較下完成？
   (a) n+Θ(logn)
   (b) 2 n
   (c) nlog n
   (d) 2n-3

3. 下列時間複雜度(time complexity) 何者的時間最少？
   (a) O (log log n)
   (b) O (logn)
   (c) O (n)
   (d) O ($\log^2$ n)

4. 以下演算法之時間複雜度分析的敘述中，n 為輸入大小，則下列敘述何者為真？
   (a) 一個0(n^2)時間的演算法一定比0(n)時間的演算法快。
   (b) 一個最佳執行時間為20 秒且最糟執行時間為100 秒的演算法的平均執行時間為60 秒。
   (c) 如果不說明，演算法的時間複雜度通常指該演算法最糟執行狀況下的情形。
   (d) 沒有任何排序演算法的時間複雜度為0(n)。

5. 下列那一個排序演算法在資料量很大的時候，其排序的速度最快？
   (a) Heap sort
   (b) Bubble sort
   (c) Insertion sort
   (d) Selection sort

6. 下列那一型態的演算法策略，當資料量很大的時候，其執行的時間複雜度將呈指數成長(exponential growth)？
   (a) Branch-and-Bound methods
   (b) Greedy methods
   (c) Dynamic programming methods
   (d) Divide-and-Conquer methods

7. 在甚麼狀況下，只能使用循序搜尋法(sequential search)而不能使用二元搜尋法(binary search)來尋找一個含有n 個元素的陣列A 中的某個元素x？
   (a) 陣列A 中的n 個元素都已排序好了。

(b) 陣列A 中的n 個元素都還沒有排序好。
(c) x 的數值是實數，無法用整數的運算來比較。
(d) 電腦CPU 速度太慢時，只能使用循序搜尋法。

8. 下列那些問題，已經存在有效率的解決的方法，也就是說，其執行的時間複雜度為多項式時間(polynomial time)？
(a) 旅行推銷員問題(traveling salesperson problem)
(b) 0/1 背包問題(0/1 knapsack problem)
(c) 凸包問題(convex hull problem)
(d) 著色問題(coloring problem)

9. 給一串由n 個不同整數所形成的數列，一個"flip"的動作定義為：將數列從開頭依序選擇前k 個數(k 為任一個大於等於1 且小於等於n 之整數)，並將此k 個數之排列順序反轉。例如給一串數列3, 4, 6, 9, 8, 2, 1, 7, 5，將其前4 個數做一次"flip"，則得到9, 6, 4, 3, 8, 2, 1, 7, 5。請問給任何一串由n 個不同數所形成的數列，假設已知最大值的所在位置，最多需要幾次"flip"的動作方可將最大值置於數列最後位置?
(a) 1 次
(b) 2 次
(c) n-1 次
(d) n 次

10. 下列的排序法中，哪一個在對7, 13, 45, 68, 91, 154, 189, 225這組資料做排序時使用的比較次數最少？
(a) 堆積排序(Heap Sort)
(b) 快速排序(Quick Sort)
(c) 插入排序(Insertion Sort)
(d) 合併排序(Merge Sort)

11. 在250筆資料當中，以二分搜尋法(Binary Search)尋找某一筆資料時，至多只要搜尋幾次即可找到？

(a) 6 次 (b) 7 次 (c) 8 次 (d) 9 次

12. 下列何種順序所建造的二元搜尋樹(Binary Search Tree)最平衡(Balanced)?
(a) 30,20,50,5,25,41,80
(b) 5,20,25,30,41,50,80
(c) 80,50,41,30,25,20,5
(d) 50,80,41,30,25,20,5

13. 給定1000筆資料儲存在陣列中，下列有關搜尋(Search)的敘述何者錯誤？
(a) 經資料排序過後，利用二分搜尋法最多只需要比較約 10 次
(b) 未排序過的資料，利用二分搜尋法最多需要比較 1000 次
(c) 如果資料是中文，經排序後，二分搜尋法最多只需要比較 10 次
(d) 排序過的資料，利用循序搜尋法最少只需要比較 1 次

14. 分別以鏈結串列(Linked List)與循序串列(Sequential List, Array)儲存10萬筆資料，下列敘述何者錯

誤？

(a) 排序過的鏈結串列利用二分搜尋法可以加速查詢

(b) 鏈結串列加入或刪除資料比循序串列所需時間少

(c) 鏈結串列需要額外的空間儲存鏈結

(d) 找第 K 大的資料，排序過的循序串列比排序過的鏈結串列所需的時間少

15. 如果依序輸入六筆資料，下列何者所建立的二元搜尋樹(Binary Search Tree)層數最少？

(a) 100, 200, 300, 400, 500, 600

(b) 300, 200, 500, 400, 100, 600

(c) 600, 500, 400, 300, 200, 100

(d) 400, 100, 500, 100, 200, 600

16. 對N個範圍在1-1000的數字排序，所需花的最少時間為何？

（a）O(n)

（b）O($n$lg n)

（c）O(1000)

（d）O(n^2)

17. 下列何者是T(n)=lg$n$+2T(n/4)這個遞迴式（Recurrence）的解？

（a）O(lg$n$)

（b）O(n)

（c）O($n^{1/2}$)

（d）O(n^2)

18. 將兩個長度為m及n的已排序數字串列合併所需花的最少時間為何？

（a）O(m+n)

（b）O((m+n)^2)

（c）O(m*n)

（d）O((m+n)lg(m+n))

19. 有關循序搜尋法（Sequential Search）的敘述何者錯誤？

（a）檔案資料未排序時才能使用

（b）搜尋時是將檔案資料一筆一筆逐一比對

（c）其演算法的時間複雜度為 O(n)

（d）在磁帶上搜尋資料一般都用此法

20. 某個問題涉及n個資料的處理，四名學生的解（演算法）皆正確，但分別需要約n^2、$n^{0.5}$、logn及n!個計算，那麼這個問題的複雜度可能為何？

（a）n^2

（b）$n^{0.5}$

（c）log n

（d）n!

21. 從五個不同數字中要找出中間值至少需要幾次比較？

（a）5 次

（b）6 次
（c）7 次
（d）8 次

22. 假設某候選員需要親訪15個鄉鎮，任兩個鄉鎮間都有專屬道路，在一次走完且任一鄉鎮不重複拜訪的條件下，從某個鄉鎮出發，共有幾種走法？
（a）14！
（b）14^{14}
（c）14^2
（d）以上皆非。

23. 雜湊法（hashing）可用來將資料表（table）中的紀錄（record）先平均打散成n群，以降低整體的搜尋時間。若n為10，則某3筆紀錄中至少兩筆會被分配到同一群的機率約為何？
（a）0.7
（b）0.6
（c）0.3
（d）0.2

24. 如欲使用二元搜尋法，則資料必須具備以下何種特性？
(1) 資料必先經過排序。
(2) 資料必須不重複。
(3) 資料必須全為正數或全為負數。
(4) 資料必須為整數。

25. 有一個數列1 3 7 10 12 24 85 按照二元樹（binary tree）的資料結構儲存，假設欲搜尋的數字為9，那麼在搜尋完成前，至少需要幾次的比較？
(1) 1
(2) 2
(3) 3
(4) 4

26. 把一個問題切割成若干個小問題，然後分別去解決個別的小問題，最後再把小問題的答案結合成大問題的解答，這樣的方法稱之為：
(1) divide and conquer
(2) dynamic programming
(3) greedy
(4) search。

27. 有25位同學彼此不認識，每天在圓桌聚餐一次，每次聚餐後每個人便會與相鄰的二人認識，則最少需要多少天後才會使得每個人皆彼此認識?
(1) 5
(2) 12
(3) 13
(4) 25

資料結構與演算法複習試題解答(出自：全國資訊競賽 89, 91、IOI 2002, 2003)

						1.	b	2.	d
3.	a	4.	d	5.	a	6.	a	7.	b
8.	b	9.	b	10.	c	11.	c	12.	a
13.	b	14.	a	15.	b	16.	a	17.	c
18.	a	19.	a	20.	c	21.	b	22.	a
23.	c	24.	1	25.	3	26.	1	27.	2