《資料結構》

試題評析	第一題：本題考的是內插法搜尋的基本特性。 第二題：本題測驗鏈結串列基本操作的做法。 第三題：本題考的是一些特殊狀況的排序選擇。 第四題：本題測驗最低成本伸展樹最重要的兩種演算法。 第五題：本題測驗資料結構的設計能力，必須利用 AVL-tree，以及一點組合能力即可，算是簡單的設計問題。 第六題：測驗考生對於有名的單起點最短路徑演算法，是否有實際完整的了解，是否清楚所需要的資料結構為何。 第七題：本題測驗考生對於程式的頻率計次計算的觀念與實際計算能力。 綜觀本次高考資料結構試題特性，有兩個主要的特點，一是題目的份量實在不算少，因此答題速度必須掌握好，如果先作答的題目耗費太多時間，可能導致答不完；二是題目有部分簡單，有部分較難些，以命題角度而言，是相當不錯的一份試題，可以鑑別出考生的程度。由此可見，未來資料結構應試者宜多加準備，才能獲取佳績。
考點命中	第一題：《資料結構》高點出版，王致強編撰，頁10-11~10-15頁，內插搜尋法。 第二題：《資料結構》高點出版，王致強編撰，頁3-16頁，將單鏈非循環串列的反轉。 第三題：《資料結構》高點出版，王致強編撰，頁9-70頁，排序綜合討論。 第四題：《資料結構》高點出版，王致強編撰，頁8-40~8-41頁，Kruskal's Algorithm 與 Prim's 演算法。 第五題：《資料結構》高點出版，王致強編撰，頁11-20~11-25頁，第11-5節 AVL-trees。 第六題：《資料結構》高點出版，王致強編撰，頁8-60~8-62頁，Dijkstra演算法。 第七題：《資料結構》高點出版，王致強編撰，頁1-27頁，精選範例14。

一、給一個排序好的陣列（Sorted Array）A[low...high]，當我們要搜尋一個元素X是否在此陣列A中，二元搜尋法（Binary Search）是檢查陣列的中間位置的元素A[next], next=[(low+high)/2]，和X做比較，並依比較結果作下列更新。

Case：

A[next]=X : return

A[next]>X : high ← next-1

A[next]<X : low ← next+1

重複上述步驟搜尋更新的陣列A[low...high]直到找到X或確認X不是在此陣列A中。若我們設計一個新的搜尋法來修改二元搜尋法，每次都是以下列方式選取A[next]。

next←low+[(high-low) * (X-A [low])/(A[high] -A[low])]

其他步驟都和二元搜尋相同。請回答下列問題：（每小題5分，共15分）

(一)新的搜尋法特色為何？請說明之。

(二)新的搜尋法在何種情形下，會比二元搜尋的搜尋速度為佳？請說明之。

(三)新的搜尋法，在最差的情況下，它的執行時間複雜度為多少？原因為何？假設陣列A中有n個元素。

答：

(一)此一新的搜尋法就是內插搜尋法，使用內插法的公式，計算要搜尋的資料X可能會在那一個位置，再加以比較，若未搜尋到，就判斷往前或是往後搜尋。

(二)在資料平均分佈時，平均搜尋時間為O(loglogn)，會比二元搜尋法的O(logn)來得好。

(三)在最差的情況下，它的執行時間複雜度為O(n)，因為當資料分佈不平均時，會退化成與線性搜尋一樣。

二、L為一鏈結串列（Linked List），函數Reverse（L）是要求把在原來L的每個節點（Node）的地址指標（Pointer），更改為指向它在鏈結串列L中的前面一個節點。請設計一個以疊代（Iterative）方式的程式來執行函數Reverse（L）的功能，程式限制只能使用常數個（constant）額外空間（External Memory），可用程式語言C、C++、Java或Pseudocode，寫出你的答案。請先說明你的作法，再寫出程式。（15分）

答：除了L之外再增加兩個指標變數，由串列開頭逐一將節點的鏈結反轉。

```
(1) NodePtr Reverse(NodePtr L)
(2) {
(3)       NodePtr p, q;
(4)       p=NULL;
(5)       while (L!=NULL)
(6)       {       q=p;   p=L;   L=L->link;     p->link=q;       }
(7)       return p;
(8) }
```

三、若只能使用下列6種方式排序（Sorting）：(a)Insertion Sort (b)Radix Sort (c)Merge Sort (d)Counting Sort (e)Heap Sort (f)Quick Sort。在下列各情形下，應選擇上述何種排序方法為最佳？請說明原因。（每小題5分，共15分）

(一)只要將全部資料中的前20名最大值排序好，並且主記憶體空間足夠。

(二)只有少數資料在被已排序好的資料修改過，需要重排序，並且主記憶體空間足夠。

(三)資料無明顯特性，需要做第一次的排序，並且主記憶體空間足夠。

答：

(一)Heap Sort：若只要排序前20名最大的資料，可以採用Heap Sort，每次從未排好的部分，選擇出最大的資料，並交換到前面的位置。在每次要選出最大的一項資料，使用Max Heap 都只需要O(logn)的時間，故總時間為最少。

(二)Insertion Sort：在插入資料時，只有少數未排序好的資料需要花O(n)的時間，其餘的每項資料都只要O(1)時間，故總時間為最節省。

(三)Quick Sort：Quick Sort 在普遍的狀況之下，平均時間為最佳的 O(nlogn)。

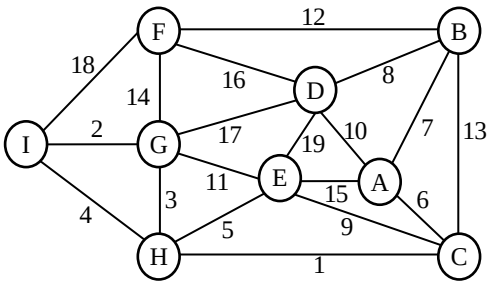
說明：Insertion Sort時間為O(n2)，複雜度較高；Radix Sort 比較適合資料很平均分佈的狀況，才能有O(kn)=O(nlogn)的時間(k為key的位數)；Merge Sort的問題則為搬動次數過多；Counting Sort 比較適合，key的範圍較小且重覆性較高的狀況；Heap Sort 的時間複雜度O(nlogn)，與Quick Sort一樣，但是Overhead 較高，會使實際執行時間比較長。故資料在沒有特定條件與特性的狀況之下，選擇Quick Sort。

四、如右的權重圖（weighted graph）共有9個節點（vertices）19條邊（edges），回答下列問題：

(一)請列出在運用Kruskal's演算法產生最小連結樹（Minimum Spanning Tree）中把邊納入最小連結樹的順序。（3分）

(二)請列出運用Prim's演算法從A點開始產生最小連結樹，把邊納入最小連結樹的順序。（4分）

(三)設計一個O(V)的演算法，判定在新增加一個(x,y)的邊到原圖形後，是否要更新已經產生的最小連結樹。（8分）

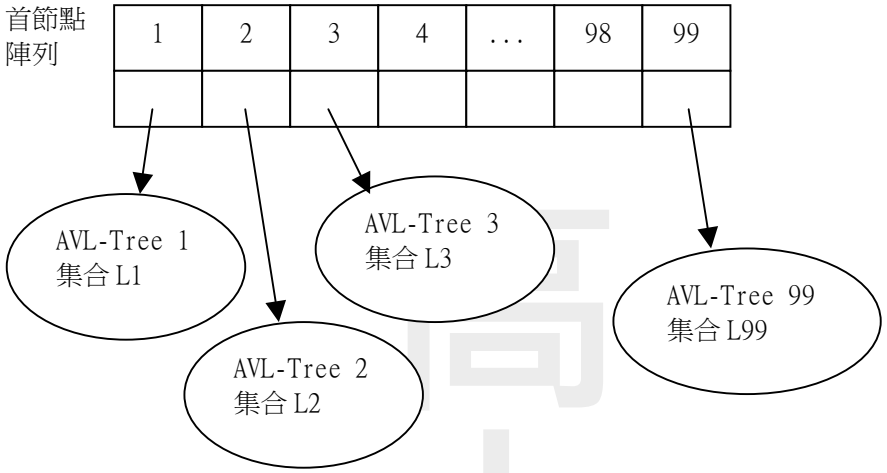


答：

(一)Kruskal's演算法的加入邊的順序依序為
(H,C),(I,G),(G,H),(E,H),(A,C),(A,B),(D,B),(B,F)
(二)Prim's演算法的加入邊的順序依序為
(A,C),(H,C),(G,H),(I,G),(E,H),(A,B),(D,B),(B,F)
(三)在連結樹上加入一個邊，必定會產生迴圈，在此一新產生的迴圈上，找出最大的一個邊，若此一最大的邊就新加入的邊，則最小連結樹與原先相同，不用更新；否則，以新加入的邊，取代掉迴圈上最大的邊，即可更新最小連結樹。因為迴圈中的邊個數≦V，故處理的時間為O(V)。

五、若處理的資料，其數值均不同且已知均為1到100之間的整數或小數。若$K \le X < K+1$，集合Lx代表數值在[K,K-1]間全部資料，$1 \le K \le 99$，K為整數，資料結構支援下列功能。
1.Insert(X)：增加X，若X不存在Lx中。
2.Delete(X)：移除X，若X存在Lx中。
3.List(X)：將Lx中的資料全部依序印出。
設計一資料結構滿足在最差情況的條件分析（Worst Case Analysis），每個功能的執行時間要求為：Insert(X) and Delete(X)須在O(log|Lx|)時間內完成，List(X)則須在O(|Lx|)時間內完成。請說明設計的資料結構為何？並解釋其執行時間為何滿足需求？（15分）

答：
(一) 使用如下圖之資料結構設計，每一個集合 Lx 都各自建立一棵AVL-tree，再使用一個首節點陣列分別指向這些 AVL-tree 的 root，這樣就可以要求。



(二)最差情況的條件分析
1.Insert(x)：由首節點陣列，找到Lx的AVL-tree root，再將資料x插入樹中，時間 O(log|Lx|)。
2.Detete(X)：由首節點陣列，找到Lx的AVL-tree root，再將x由AVL-tree 刪除，時間亦為O(log|Lx|)。
3.List(X)：要將Lx中的資料全部依序印出，也是由首節點陣列，找到 Lx 的 AVL-tree root，再對AVL-tree 進行中序走訪，時間O(|Lx|)。

六、若G=(U,E)為一權重圖（weighted graph），每條邊的權重均不為負數，則單源最短路徑問題（Single Source Shortest Path Problem）可以用著名的Dijkstra演算法求得，回答下列問題：（每小題5分，共15分）
(一)說明Dijkstra演算法的主要觀念。
(二)Dijkstra演算法在是差情況下（Worst Case Analysis），下列三個功能Insert、Delete、Decrease_Key各自需要執行的次數，可用Big-Oh符號表示。

(三)若是要在O(|E|+|V|log|V|)最差情況分析下的時間內執行Dijkstra演算法，請問該選擇使用那種資料結構，並說明其原因。

答：

(一)Dijkstra演算法的觀念：由最近的vertex開始，依照非遞減順序找出每個vertices的最短路徑。使用下列資料：

(1)cost[u,v]：邊(u,v)的成本。

(2)S為已經找到最短路徑的頂點集合。

(3)dist[w]：由起點v_0到w，目前已找到的最短路徑長度。路徑上所經過的皆為 S 中的 vertices。

(4)pred[w]：記錄由起點v_0到頂點w最短路徑上，w的直接前行者(immediate predecessor)。

演算法運算原則

(1)由V-S中選擇一個 dist 值最小的頂點u。

(2)將頂點u加入S。

(3)檢查每一個(u,w)邊，若 $w \notin S$ ，則計算

$$dist[w] \leftarrow \min\{dist[w], dist[u] + cost[u,w]\}$$

(二)Insert 需要O(|V|)次；Delete 也需要O(|V|)次；Decrease_Key需要O(|E|)次。

(三)應選擇Fibonacci-Heaps為最優，其每次 Insert 的時間為O(1)；每次Delete(Delete-min) 的時間為 O(log|V|)；而每次 Decrease_Key 的時間為O(1)。故總時間為

O(|V|)xO(1)+O(|V|)xO(log|V|)+O(|E|)xO(1)=O(|V|log|V|+|E|)。

七、下面二小題各有一段程式，其執行的時間是以執行sum++的次數計算，請用Θ-notation表示其執行時間，並說明其理由。（每小題5分，共10分）

(一)sum=0

```
for(i=0; i<2*n; i++)
  for(j=0; j<i; j++)
    sum++;
```

(二)sum=0

```
for(i=1; i<2*n; i++)
  for(j=1; j<i*i; j++)
    for(k=1; k<j; k++)
      if(j%i==1)
        sum++;
```

答：

(一)sum++ 的執行次數如下表整理

i	j	次數
0	-	0
1	0	1
2	0,1	2
3	0,1,2	3
...	...	...
2n-1	0,1,2,...,2n-2	2n-1

總計： $1+2+3+...+(2n-1)=\frac{(2n-1)(2n)}{2}=\Theta(n^2)$

(二)本題 sum++ 的執行次數與下面程式相同，使用下面程式可以讓計算簡化一些。

```
sum=0
```

```
for(i=l; i<2*n; i++)
   for(j=l; j<i*i; j++)
      if(j%i==1)
         for(k=l;k<j;k++)
            sum++;
```

故 sum++ 的執行次數如下表整理：

i	j	次數(=j-1次)	部份總和
1	-	0	0
2	1 3	0 2	2=2×(1) $=2\times\frac{2\times1}{2}$
3	1 4 7	0 3 6	3+6=3×(1+2) $=3\times\frac{3\times2}{2}$
4	1 5 9 13	0 4 8 12	4+8+12 =4×(1+2+3) $=4\times\frac{4\times3}{2}$
...	...	...	...
2n-1	1 2n 4n-1 ... (2n-1)(2n-2)+1	0 2n-1 4n-2 ... (2n-1)(2n-2)	(2n-1)+...+(2n-1)(2n-2) =(2n-1)×(1+2+3+...+(2n-2)) $=(2n-1)\times\frac{(2n-1)\times(2n-2)}{2}$

總計： $\sum_{i=0}^{2n-1}(i)\times\frac{(i)\times(i-1)}{2}=\Theta(n^4)$