109 年公務人員高等考試三級考試試題

類　　科：資訊處理
科　　目：資料結構
考試時間：2 小時

一、考慮數字 1 到 ***n***，若將其順序重新排置，每個排列順序都稱作一個排列或置換(Permutation)，例如 51432 是 12345 的一個排列。我們可以將一個數字 1 到 n 的排列視為一個順序的映射 P，則前述例子可表示為 $P(5)=1$、$P(1)=2$，$P(4)=3$、$P(3)=4$、$P(2)=5$。當然，12345 也是 12345 的一個排列。在一個數字 1 到 n 的排列 P 中，若一對數字 i 和 j，$1\le i<j\le n$，$P(j)<P(i)$，也就是在排列 P 中較大的數字 j 出現在較小的數字 i 左邊(前面)，我們稱此對數字為反向(Inversion)，而排列 P 的反向數(Inversion number) 則定義為排列 P 中反向的總數量。請回答下列問題：

㈠數字 1 到 n 的何種排列會有最大的反向數?最大反向數是多少?(5 分)

㈡若給定一個數字 1 到 n 的排列 P，請提出一個線性遞迴(Linear Recursive)的方式來算出排列 P 的反向數，並提供虛擬碼(Pseudo-code) 與時間複雜度分析。(10 分)

【解題關鍵】
《考題難易》：★★★
《破題關鍵》：本題為排序應用題，掌握反向數基本概念與合併排序運用即可得到解答。

【擬答】

㈠若數字 1 到 n 的反向排序排列(由大到小排列)會有最大反向數=(n-1)+(n-2)+...+1=n*(n-1)/2

㈡運用 merge sort 可以完成計算，程式碼如下：

```
#include <bits/stdc++.h>
using namespace std;
int _mergeSort(int arr[], int temp[], int left, int right);
int merge(int arr[], int temp[], int left, int mid, int right);
int mergeSort(int arr[], int array_size) {
   int temp[array_size];
   return _mergeSort(arr, temp, 0, array_size - 1);
}

/* An auxiliary recursive function that sorts the input array and returns the number of inversions in
the array. */
int _mergeSort(int arr[], int temp[], int left, int right) {
   int mid, inv_count = 0;
   if (right > left) {
      /* Divide the array into two parts and call _mergeSortAndCountInv()
      for each of the parts */
      mid = (right + left) / 2;

      /* Inversion count will be sum of
      inversions in left-part, right-part
      and number of inversions in merging */
      inv_count += _mergeSort(arr, temp, left, mid);
      inv_count += _mergeSort(arr, temp, mid + 1, right);
```

```
        /*Merge the two parts*/
        inv_count += merge(arr, temp, left, mid + 1, right);
    }
    return inv_count;
}

/* This funt merges two sorted arrays and returns inversion count in the arrays.*/
int merge(int arr[], int temp[], int left,
          int mid, int right)
{
    int i, j, k;
    int inv_count = 0;

    i = left; /* i is index for left subarray*/
    j = mid; /* j is index for right subarray*/
    k = left; /* k is index for resultant merged subarray*/
    while ((i <= mid - 1) && (j <= right)) {
        if (arr[i] <= arr[j]) {
            temp[k++] = arr[i++];
        }
        else {
            temp[k++] = arr[j++];

            /* this is tricky -- see above
            explanation/diagram for merge()*/
            inv_count = inv_count + (mid - i);
        }
    }

    /* Copy the remaining elements of left subarray
(if there are any) to temp*/
    while (i <= mid - 1)
        temp[k++] = arr[i++];

    /* Copy the remaining elements of right subarray
(if there are any) to temp*/
    while (j <= right)
        temp[k++] = arr[j++];

    /*Copy back the merged elements to original array*/
    for (i = left; i <= right; i++)
```

```
        arr[i] = temp[i];

    return inv_count;
}

// Driver code
int main() {
    int arr[] = { 1, 20, 6, 4, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int ans = mergeSort(arr, n);
    cout << " Number of inversions are " << ans;
    return 0;
}
```

時間複雜度分析：此演算法採用 divide and conquer，因此每一階都要走訪完整的陣列，且共有 log n 階，因此時間複雜度為 O(n log n).

二、優先佇列(Priority Queue)是依管理物件的優先權來考量，在此我們考慮管理物件的鍵值(Key)愈小其優先權愈高，兩個主要操作則分別為加入(Insert) 與擷取最小者 (Delete_Min)。

㈠請說明如何利用優先佇列對 *n* 個鍵值進行排序。(6 分)

㈡我們使用一個未排序的陣列(Unsorted Array)來管理鍵值以實現一個優先佇列，請回答下列問題：(10 分)

⑴若有 n 個鍵值，請說明兩個主要操作(加入(Insert)與擷取最小者(Delete_Min))的時間複雜度。

⑵請判斷下面的敘述是否為真，並請說明原因：

若以此優先佇列進行排序(Sorting)，其所對應的排序原理為插入排序 (Insertion Sort) 。

㈢二元堆積 (Binary Heap)是一個優先佇列的資料結構，因為我們考慮鍵值小的物件有高的優先權，所以又可稱為最小堆積(Minimum Heap)。(14 分)

⑴在結構上最小堆積為一個完全二元樹(Complete Binary Tree)，若使用一個陣列來實作最小堆積，陣列中物件的鍵值放置如下，請描述此陣列對應的完全二元樹(以樹狀結構表示)。

Index	1	2	3	4	5	6	7	8	9	10
Key	35	18	42	24	7	14	25	12	38	21

⑵請說明二元堆積中何謂堆積特性 (Heap Property) ?

⑶前揭⑴中的完全二元樹並未有堆積特性，請將其進行堆積化(Heapify)，並以陣列表示出堆積化後的最小堆積所對應之完全二元樹

【解題關鍵】
《考題難易》：★★
《破題關鍵》： 本題為運用堆積處理優先佇列基本題，掌握堆積觀念與相關操作即可得到解答。

【擬答】

㈠只要先用加入操作將欲排序鍵值一個個加入優先佇列，接著用擷取最小者從優先佇列中一個個刪除，即可得到由小到大排序。

㈡⑴若用二元最小堆積來實現優先佇列，則此時

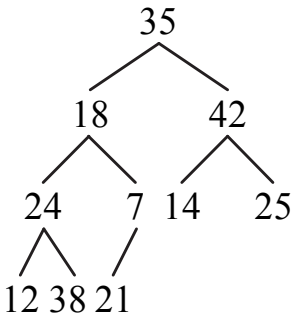
A. 加入(Insert)： 我們在樹的末尾添加一個新鍵值。 如果新鍵值大於其父鍵值，則我們無需執行任何操作。 否則，我們需要往上走訪以修復違反的二元堆積屬性，最多需要

進行 log n 次，因此插入新鍵值需要 0（logn）時間。

B. 擷取最小者(Delete_Min)：從 MinHeap 中刪除最小元素。 此操作的時間複雜度為 0（logn），因為此操作需要在除去根之後通過呼叫 heapify（）維護堆積屬性。

⑵應為對應 Heap sort，而非對應插入排序。

㈢⑴此陣列對應的完全二元樹(以樹狀結構表示)如下：



⑵最小堆積的堆積特性 (Heap Property)就是所有節點的值恆小於等於子節點的值，且仍需維持 Complete Binary Tree。

⑶以陣列表示出堆積化後的最小堆積所對應之完全二元樹為

Index	1	2	3	4	5	6	7	8	9	10
Key	7	12	14	24	18	42	25	35	38	21

三、請回答下列關於 AVL 樹(AVL Tree)的問題:

㈠我們欲將所管理的鍵值(Key)依序列出，請問是否可以利用一個 AVL 樹對鍵值來進行排序(Sorting)?若不行，請說明原因;如果可以，請描述方法及時間複雜度。(5 分)

㈡請提供一個線性時間的演算法來判斷一個二元搜尋樹是否為 AVL 樹。(10 分)

㈢在 AVL 樹上進行一個加入(Inset)操作後，是否最多只需要一次的重構(Restructuring)即可恢復其平衡的特性?請說明原因。(10 分)

【解題關鍵】
《考題難易》：★★★
《破題關鍵》： 本題為 AVL 樹應用題，掌握 AVL 樹運作方式即可得到解答。

【擬答】

㈠因為 AVL 樹為接近平衡的二元搜尋樹，因此只要對其進行中序走訪，即可獲得由小到大排序的結果，時間複雜度為 O($n$log n)。

㈡

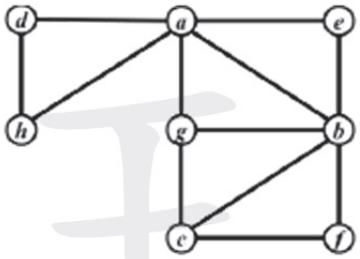
```
bool isBalanced(node* root, int* height) {
  /* lh --> Height of left subtree   rh --> Height of right subtree */
  int lh = 0, rh = 0;
  /* l will be true if left subtree is balanced and r will be true if right subtree is balanced */
  int l = 0, r = 0;
   if (root == NULL) {
    *height = 0;
    return 1;
  }
```

```
    /* Get the heights of left and right subtrees in lh and rh And store the returned values in l and r
  */
    l = isBalanced(root->left, &lh);
    r = isBalanced(root->right, &rh);

    /* Height of current node is max of heights of left and right subtrees plus 1*/
    *height = (lh > rh ? lh : rh) + 1;

    /* If difference between heights of left and right subtrees is more than 2 then this node is not
  balanced so return 0 */
    if (abs(lh -rh) >= 2)
      return 0;

    /* If this node is balanced and left and right subtrees are balanced then return true */
    else
      return l && r;
  }
```

此一演算法由於在同一次遞迴中計算左右子樹高度，因此時間複雜度為 O(n)

㈢在 AVL 樹插入節點後依照插入節點與失去平衡節點的相對關係可分成 LL、RR、LR、RL 等四種情況，在 LL 和 RR 的情況下，只需要進行一次旋轉操作；在 LR 和 RL 的情況下，需要進行兩次旋轉操作。旋轉完成(一次重構)後即可恢復高度平衡，因為此時已將失去平衡節點平衡因子調整，因此插入操作不需再次重構。

四、若我們用相鄰矩陣(Adjacency Matix) M 來表示圖一中的無向圖 G=(V，E)，請考慮下面的問題：



圖一、無向圖 $G=(V, E)$

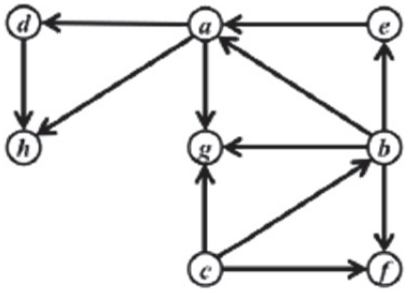
㈠對於無向圖 $G=(V,E)$:(12 分)

⑴請給出對應的相鄰矩陣 M。

⑵以字母順序為考量進行深度優先搜尋(Depth-First Search， DFS)，請由節點 a 開始，描述此深度優先搜尋所產生的深度優先樹(DF-tree)。

㈡請說明在用相鄰矩陣(Adjacency Matrix)表示的無向圖上，進行深度優先搜尋的時間複雜度，其中節點與邊的數量分別為$|V|=n$ 與$|E|=m$。(8 分)

㈢若將圖一無向圖 $G=(V,E)$中的邊給予方向成為如圖二中的有向圖(Directed Graph) G'：(10 分)



圖二、有向圖G

⑴有向圖 G'沒有迴圈(Cycle)，是一個無迴圈有向圖(Directed Acyclic Graph， DAG)，所以存在節點的拓樸排序(Topological Sort)，請對 G'給出一個拓樸排序(Topological Sort)。

⑵請給一個方法來判斷一個有向圖是否沒有迴圈。

【解題關鍵】
《考題難易》：★★
《破題關鍵》：
本題為圖形基本題，掌握圖形相關的相關矩陣表示、DFS、拓樸排序、迴圈判斷等傳統考題即可得到解答。

【擬答】

㈠

⑴相鄰矩陣表示如下

	a	b	c	d	e	f	g	h
a	0	1	0	1	0	0	1	1
b	1	0	1	0	1	1	1	0
c	0	1	0	0	0	1	1	0
d	1	0	0	0	0	0	0	1
e	1	1		0	0	0	0	0
f	0	1	1	0	0	0	0	0
g	1	1	1	0	0	0	0	0
h	1	0	0	1	0	0	0	0

⑵以字母順序為考量進行深度優先搜尋，過程如下：

①由 a 出發，其鄰居有 bdegh 均未走訪，依照字母順序前往 b

② b 的鄰居有 acefg，其中 cefg 未走訪，依照字母順序前往 c

③ c 的鄰居有 bfg，其中 fg 未走訪，依照字母順序前往 f

④ f 的鄰居有 bc，均已走訪，故退回 c，依照字母順序前往未走訪的 g

⑤ g 的鄰居有 abc，均已走訪，故退回 c，但 c 的鄰居均已走訪；再退回 b，依照字母順序前往未走訪的 e

⑥ e 的鄰居有 ab，均已走訪，故退回 b，但 b 的鄰居均已走訪；再退回 a，依照字母順序前往未走訪的 d

⑦ d 的鄰居有 ah，其中 h 未走訪，前往走訪 h，完成走訪

整個走訪順序為 a->b->c->f->g->e->d

㈡使用鄰接矩陣，找到所有輸出邊所需的時間為 O（n），因為必須檢查節點行中的所有 n 列。匯總所有 n 個節點，得出 O（n^2）。

㈢⑴其中一個拓樸排序為

①找出沒有 in-edge 的節點，此時僅有 c，接著刪除此節點與其 out-edge

②找出沒有 in-edge 的節點，此時僅有 b，接著刪除此節點與其 out-edge

③找出沒有 in-edge 的節點，此時有 ef，設選 f，接著刪除此節點與其 out-edge
④找出沒有 in-edge 的節點，此時僅有 e，接著刪除此節點與其 out-edge
⑤找出沒有 in-edge 的節點，此時僅有 a，接著刪除此節點與其 out-edge
⑥找出沒有 in-edge 的節點，此時有 dg，設選 g，接著刪除此節點與其 out-edge
⑦找出沒有 in-edge 的節點，此時僅有 d，接著刪除此節點與其 out-edge
⑧找出沒有 in-edge 的節點，此時僅有 h，接著刪除此節點，已經完成拓樸排序
最終過程為 cbfeagdh

⑵①使用給定數量的邊和頂點創建圖形。
②創建一個遞迴函數，以初始化當前索引或頂點，已訪問和遞迴堆疊。
③將當前節點標記為已訪問，並在遞迴堆疊中標記索引。
④查找所有未訪問且與當前節點相鄰的頂點。 遞迴呼叫這些頂點的函數，如果遞迴函數返回 true，則返回 true。
⑤如果在遞迴堆疊中已經標記了相鄰的頂點，則返回 true。
⑥創建一個包裝器類別，該包裝器為所有頂點呼叫遞迴函數，如果有任何函數返回 true，則返回 true。 否則，如果函數對於所有頂點都返回 false，則返回 false。