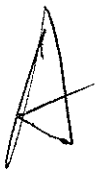
Advanced Graph Algorithms and Applications

Homework 1

資工碩 1 79883107 蔡維峻



壹、 資料結構

變數	說明
n	蘋果數量
A_i	Apple，Input 資料
W_{Max}	最重的蘋果重量
W_{min}	最輕的蘋果重量
d	W_{Max} 與 W_{min} 的差距
C_i	Count，記錄每一種重量的蘋果數量
S_i	Sort，排序好之後儲存的一維陣列

貳、 演算法

假設蘋果重量皆為整數。

Algorithm:

1. Begin
2. W_{Max} ← the weight of the biggest apple
3. W_{min} ← the weight of the smallest apple
4. d ← (W_{Max} - W_{min})

5. $C_{0\sim d} \leftarrow 0$

6. for i = 1 to n

7. $C[A_i\text{-}W_{min}] \leftarrow C[A_i\text{-}W_{min}] + 1$

8. loop
9. for i = 1 to d

10. $C[i] \leftarrow C[i] + C[i\text{-}1]$

11. loop
12. for i = 1 to n

13. $S[C[A_i\text{-}W_{min}]] \leftarrow A_i$

14. $C[A_i\text{-}W_{min}] \leftarrow C[A_i\text{-}W_{min}] - 1$

15. loop
16. End

參、 實例

A

50	52	53	55	53	61	59	63	61	55	55	48	60	61	51

$\therefore W_{max} = 63$, $W_{min} = 48$ and $d = W_{max} - W_{min} = 15$

C(1)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1	0	1	1	1	2	0	3	0	0	0	1	1	3	0	1

C(2)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1	1	2	3	4	6	6	9	9	9	9	10	11	14	14	15

S

	50													
	50		52											
	50		52		53									
	50		52		53			55						
	50		52	53	53			55						
	50		52	53	53			55					61	
	50		52	53	53			55	59				61	
	50		52	53	53			55	59				61	63
	50		52	53	53		55	55	59			61	61	63
	50		52	53	53	55	55	55	59			61	61	63
48	50		52	53	53	55	55	55	59			61	61	63
48	50		52	53	53	55	55	55	59	60		61	61	63
48	50		52	53	53	55	55	55	59	60	61	61	61	63
48	50	51	52	53	53	55	55	55	59	60	61	61	61	63

肆、 效能分析

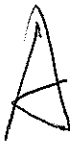
找出 W_{Max} 和 W_{min} 皆花費 O(n)，記錄各重量之蘋果數量也花費 O(n)，將 C_i 各元素作運算需花費 O(d)，Sorting 則花費 O(n)，故時間複雜度為 O(d+n)，空間複雜度也為 O(d+n)

79882301 周德智 Advanced Graph Algorithms and Applications Hw1

想要把 N 個蘋果依重量分成三類，重量最重的打算兜售，重量次重的打算送人，重量輕的打算自己食用。

為了避免把蘋果撞爛，先把蘋果的重量測量出來然後編號。這問題可以對應到在一個未排序的數列如果找到第 N/3、2N/3 大的數來對數列進行分類。以下有兩種方法來討論如何來把數列分成三等份。所以資料結構的部分都是使用 1 個 array，空間複雜度為 Θ(N)。

方法一 利用找出第 K 大的數的方法來找出第 N/3 大的數



思考找出第 K 大的數需要花多少時間？
找出第 1 大的數需要比較 N 次
再找出第 2 大的數需要比較 N-1 次
...
再找出第 K 的數需要比較 N-(K-1)次
可以知道從 N 個數要找出第 K 大的數總共需要 N+(N-1)+...N-(K-1)=K(2N-K+1)/2 次
所以要找出第 N/3 大的數需要把 K=N/3 代入，可以得到需要比較 N(5N+3)/18 次
然後利用第 N/3 大的數先把數列分成兩群，一群是第 1~N/3 大的數，另外一群是第 N/3~N 大的數，這需要比較 N 次。
然後再用一樣的方法在第 N/3~N 大的數這群中找出第 2N/3 大的數，可以得到需要比較 N(N+1)/6 次，然後利用也一樣分成兩群，這需要比較 2N/3 次。
所以總共需要比較 N(4N+18)/9 次。

方法二 利用 Randomized-Quick Sort 來把數列排序，可以預期所需要的時間複雜度大約在 $O(N*Log_2N)$，再把蘋果分類到所屬的群組裡，時間是 O(N)，所以總共的時間複雜度會在 $O(N*Log_2N)$。

	時間複雜度	空間複雜度
方法一**(Selection Sort)**	$O(N^2)$	O(N)
方法二**(Quick Sort)**	$O(N*Log_2N)$	O(N)

方法一的時間複雜度是來自 O(K*N)，所以如果要找出前面 K 重的蘋果，而且 $K<Log_2N$，使用方法一是比較好的，不過如果在一般情況下要將蘋果依照重量等量分成等量的三等份我想還是使用 Quick Sort 比較合適。

如果可以不用精確的分成三等份，或許可以考慮先將蘋果總重量求出來再依照過去的統計數據選擇一個範圍來將蘋果分類。

國 立 台 北 大 學
電 機 工 程 研 究 所

Advanced Graph Algorithms and Applications

Homework - 1

A-

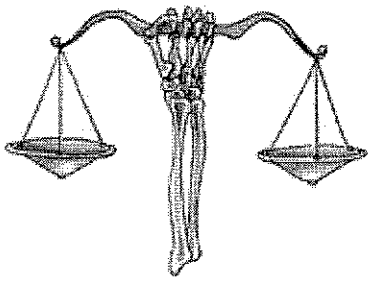
研 究 生:尤莉雅

學　　號:9882306

中 華 民 國 九 十 八 年 十 月 十 四 日

Homework Assignment No. 1

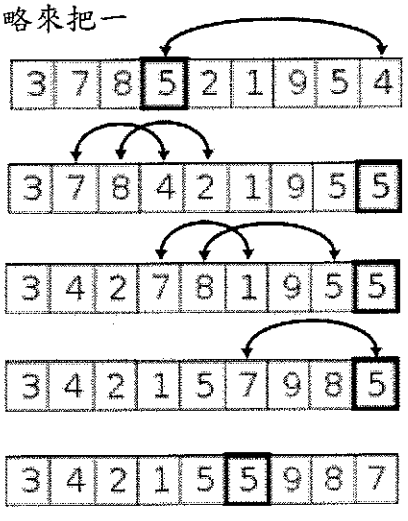
有一天有一位農夫採收一堆蘋果，他想要把蘋果依造重量分成三類，重量最重的打算兜售，重量次重的打算送人，重量輕的打算自己食用，他現在正在考慮該用哪種資料結構及演算法，請各位幫他想想辦法吧。
(請各位打成報告的模式，使用哪種演算法(不限定)?哪種資料結構(不限定)?為什麼?最後再舉個範例,最好可以分析時間空間複雜度)



ANS:

1. 使用哪種演算法(不限定)?

快速排序法是使用分治法（Divide and conquer）策略來把一個序列（list）分為兩個子序列（sub-lists）。



步驟為：

1. 從數列中挑出一個元素，稱為 "基準"（pivot），
2. 重新排序數列，所有元素比基準值小的擺放在基準前面，所有元素比基準值大的擺在基準的後面（相同的數可以到任一邊）。在這個分割之後，該基準是它的最後位置。這個稱為分割（partition）操作。
3. 遞歸地（recursive）把小於基準值元素的子數列和大於基準值元素的子數列排序。

遞迴的最底部情形，是數列的大小是零或一，也就是永遠都已經被排序好了。雖然一直遞迴下去，但是這個演算法總會結束，因為在每次的迭代（iteration）中，它至少會把一個元素擺到它最後的位置去。

快速排序法的觀念是將待排序的 N 個鍵值分成左右兩半，左半邊之鍵值小於第一個鍵值，而右半邊則大於或等於第一個鍵值。由小到大排序完之後再依重量分成輕中重三堆。

```
#include <functional>
#include <algorithm>
#include <iterator>

template< typename BidirectionalIterator, typename Compare >
void quick_sort( BidirectionalIterator first, BidirectionalIterator last, Compare cmp )
{
  if( first != last ) {
    BidirectionalIterator left  = first;
    BidirectionalIterator right = last;
    BidirectionalIterator pivot = left++;

    while( left != right ) {
      if( cmp( *left, *pivot ) ) {
        ++left;
      } else {
        while( (left != right) && cmp( *pivot, *right ) )
          right--;
        std::iter_swap( left, right );
      }
    }

    if cmp( *pivot, *left )
        --left;
    std::iter_swap( first, left );

    quick_sort( first, left, cmp );
    quick_sort( right, last, cmp );
  }
}

template< typename BidirectionalIterator >
inline void quick_sort( BidirectionalIterator first, BidirectionalIterator last ) {
  quick_sort( first, last,
    std::less_equal< typename std::iterator_traits< BidirectionalIterator
>::value_type >()
  );
}
```

2.哪種資料結構(不限定)?

陣列(array)

3.為什麼?

為什麼使用快速排序法?因為快速排序（quick sort）又稱為劃分交換排序（partition exchange sorting)。就平均時間而言，快速排序是所有內部排序中最佳的。

在做對調位置時，陣列方式使用 link list 需要不停地從 Head 尋找被更換的鍵值 O(n)。使用陣列只需在 O(1)時間達成使用陣列的好處是減少記憶體使用量，雖然額外使用了一個索引陣列，但對於頂點越多時，記憶體的減少使用會更顯著，但缺點就是必須額外耗用一些運算是處理頂點資訊。

4.最後再舉個範例,最好可以分析時間空間複雜度

(1)範例：

假設有 n 個 R1, R2, R3, ..., Rn，其鍵值為 K1, K2, K3, ..., Kn。快速排序法其步驟如下：

1.以第一個記錄的鍵值k1做基準K。
2.由左至右i = 2,3,...,n，一直找到ki> K。
3.由右至左j = n, n-1, n-2, ..., 2，一直找到kj< K
4.當i<j 時Ri與Rj互換，否則R1與Rj互換。

例如有十個記錄，其鍵值分別為39, 11, 48, 5, 77, 18, 70, 25, 55, 33，利用快速排序過程如下：

R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	
㊴	11	48	5	77	18	70	25	55	33	∵i<j ∴R_3與R_{10}對調
		i							j	
㊴	11	33	5	77	18	70	25	55	48	∵i<j ∴R_5與R_8對調
				i			j			
㊴	11	33	5	25	18	70	77	55	48	∵i>j ∴R_1與R_6對調
					j	i				
[18	11	33	5	25]	39	[70	77	55	48]	

此時在39的左半部之鍵值皆比39小，而右半部皆比39大。再利用上述方法將左半部與右半部排序，形成遞迴（recursive）。全部排序過程如下所示：

R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}
39	11	48	5	77	18	70	25	55	33
[18	11	33	5	25]	39	[70	77	55	48]
[5	11]	18	[33	25]	39	[70	77	55	48]
5	11	18	[33	25]	39	[70	77	55	48]
5	11	18	25	33	39	[70	77	55	48]
5	11	18	25	33	39	[55	48]	70	[77]
5	11	18	25	33	39	48	55	70	[77]
5	11	18	25	33	39	48	55	70	77

快速排序是unstable，最壞時間是O(n2)，平均時間是O(n log2n) 。

(2)時間複雜度：

worst case 為 $O(n^2)$

average case 為 $O(n\log n)$

best case($n\log n$)

(3)空間複雜度：

被快速排序所使用的空間，依照使用的版本而定。

使用原地（in-place）分割的快速排序版本，在任何遞迴呼叫前，僅會使用固定的額外空間。然而，如果需要產生 O(log n)巢狀遞迴呼叫，它需要在他們每一個儲存一個固定數量的資訊。因為最好的情況最多需要 O(log n)次的巢狀遞迴呼叫，所以它需要 O(log n)的空間。最壞情況下需要 O(n)次巢狀遞迴呼叫，因此需要 O(n)的空間。

如果我們考慮排序任意很長的數列，我們必須要記住我們的變數像是 left 和 right，不再被認為是佔據固定的空間；也需要 O(log n)對原來一個 n 項的數列作索引。

因為我們在每一個堆疊框架中都有像這些的變數，實際上快速排序在最好跟平均的情況下，需要 0(log2 n)空間的位元數，以及最壞情況下 0(n log n)的空間。然而，這並不會太可怕，因為如果一個數列大部份都是不同的元素，那麼數列本身也會佔據 0(n log n)的空間位元組。

非原地版本的快速排序，在它的任何遞迴呼叫前需要使用 0(n)空間。在最好的情況下，它的空間仍然限制在 0(n)，因為遞迴的每一階中，使用與上一次所使用最多空間的一半，且$\sum_{i=0}^{\infty} \frac{n}{2^i} = 2n.$ 它的最壞情況是很恐怖的，需要

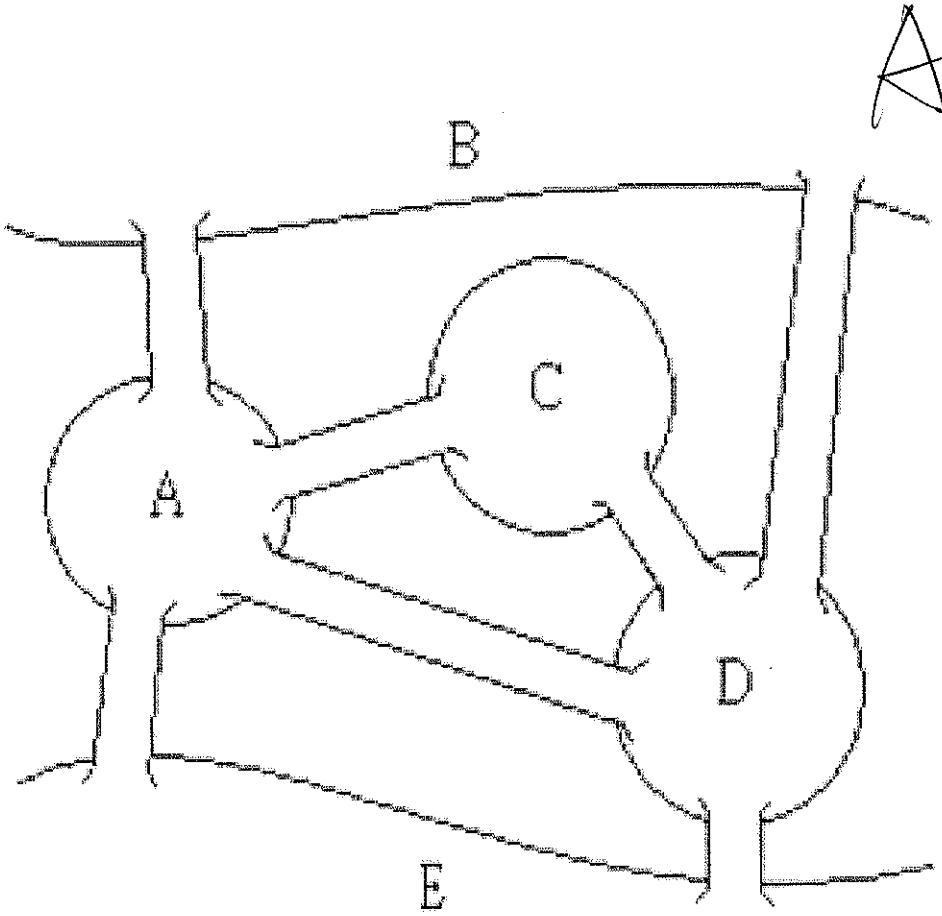
$\sum_{i=0}^{n}(n-i+1) = \Theta(n^2)$ 空間，遠比數列本身還多。

如果這些數列元素本身自己不是固定的大小，這個問題會變得更大；

舉例來說，如果數列元素的大部份都是不同的，每一個將會需要大約 0(log n)為原來儲存，導致最好情況是 0(n log n)和最壞情況是 0(n2 log n)的空間需求。

進階圖形演算法及其應用

Advanced Graph Algorithms and Applications



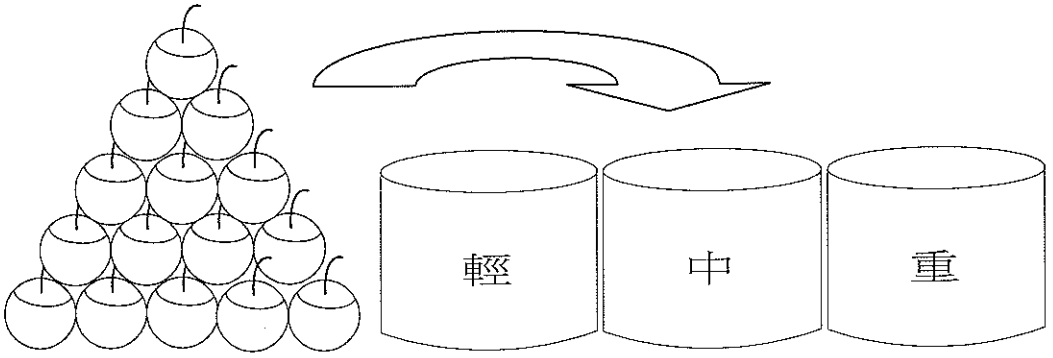
班級：產碩專班

姓名：蔡宗成

學號：**79780109**

Homework Assignment No. 1

有一天有一位農夫採收一堆蘋果，他想要把蘋果依造重量分成三類，重量最重的打算兜售，重量次重的打算送人，重量輕的打算自己食用，他現在正在考慮該用哪種資料結構及演算法，請各位幫他想想辦法吧。
(請各位打成報告的模式，使用哪種演算法(不限定)?哪種資料結構(不限定)?為什麼?最後再舉個範例,最好可以分析時間空間複雜度)



ANS:

一、使用哪種演算法(不限定)?

使用快速排序法(Quick sort)，其演算法如下；

```
void  quicksort ( element  list[ ] , int left , int right )
{
    int  pivot , i , j ;
    element  temp ;
    if  ( left < right )  {
        i = left ; j = right + 1 ;
        pivot = list[left].key ;
        do  {
            do
                i ++ ;
            while  ( list [ i ].key < pivot ) ;
            do
                j -- ;
            while ( list [ j ].key > pivot ) ;
            if  ( i < j )
                SWAP ( list[i] , list[j] , temp )
        }  while ( i < j ) ;
        SWAP ( list[left] , list[j] , temp ) ;
        quicksort ( list , left , n-1 ) ;
        quicksort ( list , j+1 , right ) ;
    }
}
```

快速排序法(Quick sort)的觀念是將待排序的N個鍵值分成左右兩半，左半邊之鍵值小於第一個鍵值，而右半邊則大於或等於第一個鍵值。

作法：

1、取序列中第一個記錄的鍵值 K 當做 *pivot*。

2、由左而右，$i=1,2,....$，找到第一個 $K_i \geq K$。

由右而左，$j=n,n\text{-}1,....$，找到第一個 $K_j \leq K$。

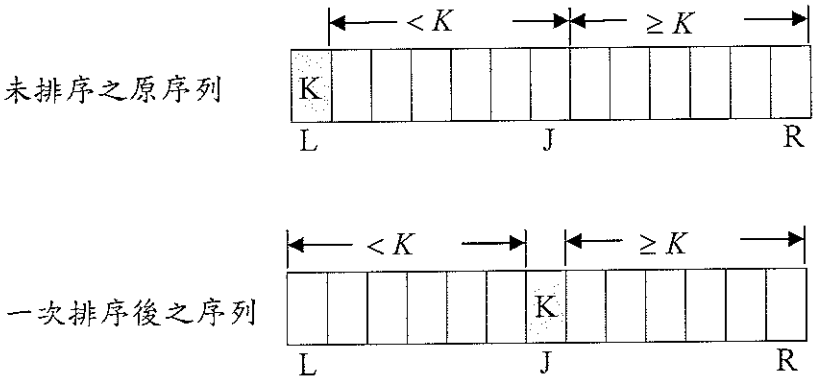
3、若 $i<j$ 則 R_i 與 R_j 對調位置。

否則，R 與 R_j 對調位置，將此序列一分為二：

$$(R_1,R_2,\cdots,R_{i-1}),R_j,(R_{j+1},R_{j+2},\cdots,R_n)$$

接著將兩個子序列獨立的重覆進行1到3步驟。

圖示：



說明：

將所有蘋果隨意排成一列，利用快速排序(Quick sort)演算法將蘋果從最輕到最重依序排列完成。如此一來，N個蘋果中，前 $N/3$ 堆為重量輕自己食用的蘋果，中間 $N/3$ 堆為次重送人用的蘋果，最後 $N/3$ 堆為最重兜售用的蘋果。

二、哪種資料結構(不限定)?

陣列(Array)

三、為什麼?

1. 為什麼使用快速排序法?

在考慮平均執行時間的前提下，快速排序是內部排序最好方法，時間複雜度為 $O(n\log n)$。

2. 為什麼使用陣列(Array)?

(1)Link list 每次讀取一筆資料時，還需花費另外時間去取出鏈結指標，以便讀取下一資料。

(2)Link list 需額外花費空間來儲存鏈結，比 Array 浪費空間(以單一節點而言)。

(3)由於 Quick sort 會從序列最左邊以及最右邊開始尋找大於或小於 *pivot* 的鍵值，假如使用 Link list 資料結構，必須製作成雙向鏈結，比一般 Link list 更浪費空間。

(4)Array 可以隨意讀取第 i 個節點，而 Link list 不能。

四、最後再舉個範例, 最好可以分析時間空間複雜度

1. 範例：

一群新兵剛到部隊報到，分隊長想要依照身高將新兵排成五個分隊，請問要怎麼安排？

Ans:

先將新兵隨意排成一列，利用快速排序(Quick sort)演算法將新兵依照身高排列好，再依照人數總數便可以很快的分成五個分隊。

2. 時間複雜度：

快速排序法(Quick sort)：

worst case 為 $O(n^2)$

average case 為 $O(n\log n)$

best case 為 $O(n\log n)$

在此題目中，由於最後還要將蘋果分成三堆，所以時間複雜度都要再加上 n

worst case 為 $O(n^2+n)\approx O(n^2)$

average case 為 $O(n\log n+n)\approx O(n\log n)$

best case 為 $O(n\log n+n)\approx O(n\log n)$

3.空間複雜度：

假設 S(n)為 n 個元素用 QuickSort 所需的空間，每次遞迴呼叫需要的空間為 c，亦即 T(1)=c。

狀況 1 (最佳狀況)：每次都剛好分成相等的 2 堆

$S(n)=S(n/2)+1$ (呼叫到下一層，故加 1)

展開得：

$$S(n)=S(n/2)+1=S(n/4)+2=S(n/8)+3=\cdots=S(n/2^k)+k$$

假設 $n=2^k$，則

$S(n) = S(1)+k=c+\log n$

故需要 $O(n\log n)$ 空間

狀況 2 (最差狀況)：每次都剛好分成 1 和 n-1 的 2 堆

$S(n)=S(n-1)+1$ (以資料最多的那個呼叫為準)

展開得：

$$S(n)=S(n-1)+1=S(n-2)+2=S(n-3)+3=\cdots=S(1)+n-1=c+n-1$$

故需要 $O(n)$ 空間

實作上的考慮：

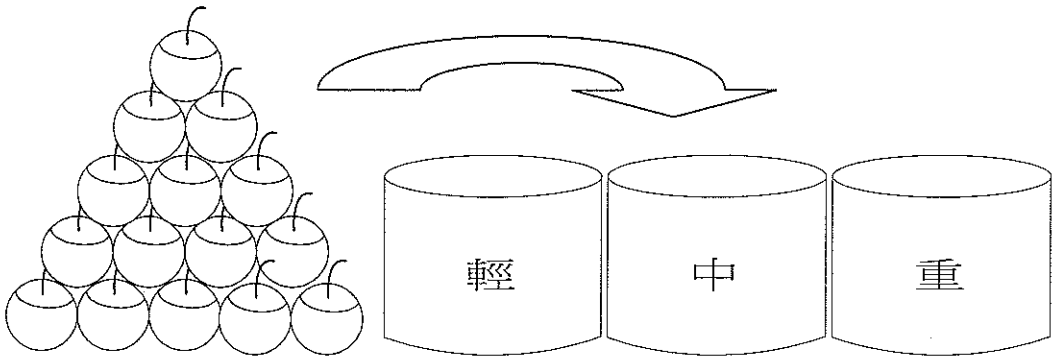
事實上只需要 $O(\log n)$ 空間。

主要是先以遞迴呼叫處理資料最少的那堆，然後重設參數，用迴圈來重複處理資料最多的那堆。如此遞迴呼叫，會處理的資料最多是 n/2，和狀況 1 相同，故最差的狀況是 $O(\log n)$。原來最差的狀況反而變成最佳狀況，所需的空間只有 O(1)。

產碩專班 陳嘉鴻 79780102

Homework Assignment No. 1

有一天有一位農夫採收一堆蘋果，他想要把蘋果依造重量分成三類，重量最重的打算兜售，重量次重的打算送人，重量輕的打算自己食用，他現在正在考慮該用哪種資料結構及演算法，請各位幫他想想辦法吧。
(請各位打成報告的模式，使用哪種演算法(不限定)?哪種資料結構(不限定)?為什麼?最後再舉個範例,最好可以分析時間空間複雜度)



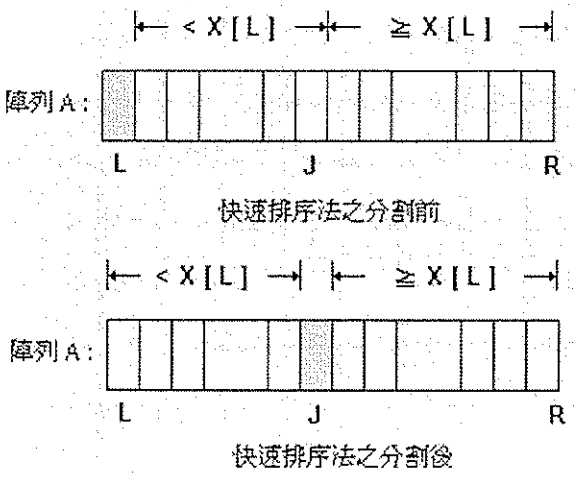
BT

ANS:

1.使用哪種演算法(不限定)?

快速排序法(Quick sort)

快速排序法的觀念是將待排序的 N 個鍵值分成左右兩半，左半邊之鍵值小於第一個鍵值，而右半邊則大於或等於第一個鍵值。



作法：

1、取第一個記錄的鍵值 K 當做 Control Key 。

2、由左而右，i = 1 , 2 , ... , 找到第一個 Ki ≧K。

由右而左，j = n , n - 1 , ... , 1 , 找到第一個 Kj ≦ K。

3、若 i < j 則 Ri 與 Rj 對調位置。

否則，R 與 Rj 對調位置，將此檔案一分為二

(R1 , R2 , R3 , ... , Ri-1) 、Rj (Rj+1 , Rj+2 , ... , Rn)，

如此兩個子檔案可獨立的重覆 1 到 3 進行。

由小到大排序完之後再依重量分成輕中重三堆

2.哪種資料結構(不限定)?

陣列(array)

3.為什麼?

(1)為什麼使用快速排序法?

在考慮平均執行時間的前題下，快速排序是內部排序最好方法。

(2)為什麼使用陣列?

在做對調位置時，使用 link list 需要不停地從 Head 尋找被更換的鍵值 O(n)。

使用陣列只需在 O(1)時間達成

4.最後再舉個範例,最好可以分析時間空間複雜度

(1)範例：

(2)時間複雜度：

worst case 為 $O(n^2)$

average case 為 $O(n\log n)$

best case($n\log n$)

(3)空間複雜度：

要看寫法，先說明 2 次都用遞迴呼叫運算的狀況. 定義如下:

假設 S(n)為 n 個元素用 QuickSort 所需的空間, 每次遞迴呼叫需要的空間為 c,

亦即 T(1)=c.

==========

狀況 1 (最佳狀況): 每次都剛好分成相等的 2 堆

S(n) = S(n/2)+1 (呼叫到下一層,故加 1)

展開得:

$S(n) = S(n/2)+1 = S(n/4)+2 = S(n/8)+3 = \ldots = S(n/2^k) + k$

假設 $n=2^k$, 則

S(n) = S(1) + k = c + logn

故需 O(logn)空間

==========

狀況 2 (最差狀況). 每次都剛好分成 1 和 n-1 的 2 堆

S(n) = S(n-1)+1 (以資料最多的那個呼叫為準)

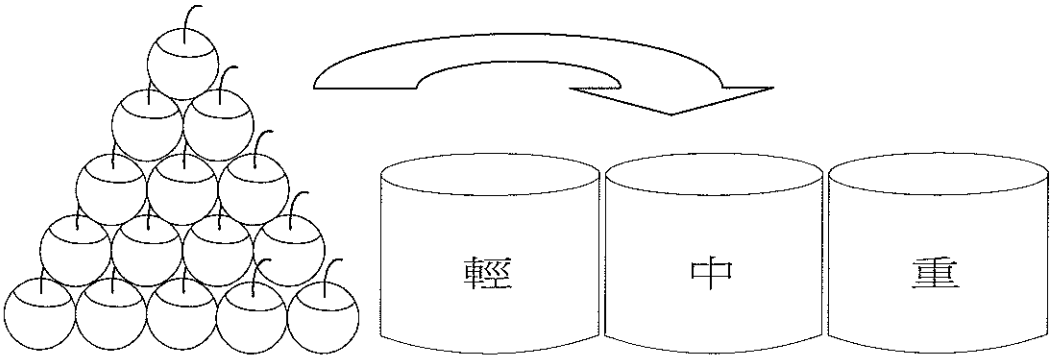
展開得:

$S(n) = S(n-1)+1 = S(n-2)+2 = S(n-3)+3 = \ldots = S(1)+n-1 = c+n-1$

故需 O(n)空間

有一天有一位農夫採收一堆蘋果，他想要把蘋果依造重量分成三類，重量最重的打算兜售，重量次重的打算送人，重量輕的打算自己食用，他現在正在考慮該用哪種資料結構及演算法，請各位幫他想想辦法吧。

(請各位打成報告的模式，使用哪種演算法(不限定)?哪種資料結構(不限定)?為什麼?最後再舉個範例，最好可以分析時間空間複雜度)



B+

Answer：

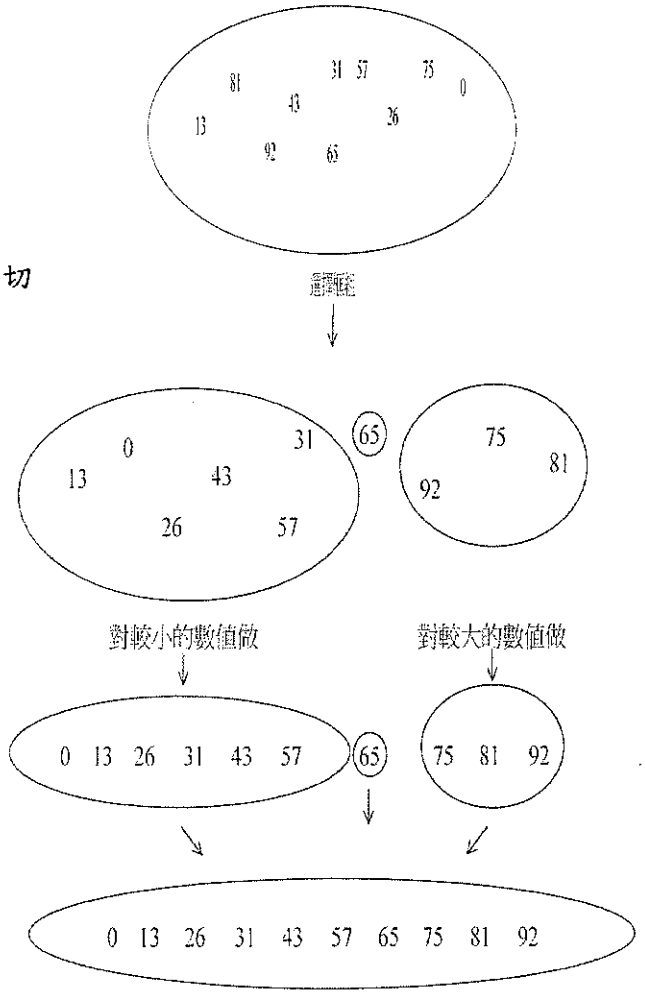
1. 使用哪種演算法(不限定)?

1.1 quick sort 演算法
(以右圖為例，將選擇框內作排序)

1.2 Step1：隨選數值為 pivot(例 65)，並作切割動作。

Step2：比 65 大的值放入右邊 set，比 65 小的值放入左邊 set。

Step3：將左右 set 重複上述動作，Quick sort 即可完成。



2. 哪種資料結構(不限定)?

以陣列來 implement 比較省空間。

3. 為什麼?

3.1 這種分類問題，最好選擇以 divided&conquer 的 sorting 來 implement。

3.2 作數值調換動作時，使用 array 只耗費時間 O(1)完成，

若使用 linked-list 要從 head 搜尋到 rear，耗費 O(n)時間。

4. 時間複雜度

4.1 基本的 quick sort 關係，$T(N) = T(i) + T(N - i - 1) + cN$

4.2 在 worst-case 時，pivot 為最小的元素，假設忽略 T(0)=1，$T(N) = T(N - 1) + cN, N > 1$，可得

$$T(N) = T(1) + c\sum_{i=2}^{N} i = O(N^2)$$

4.3 在 Best-Case 時，pivot 是在中間，則 $T(N) = 2T(N/2) + cN$，產生 $T(N) = cN \log N + N = O(N \log N)$

在 average-case 時，$T(N) = O(N \log N)$

5. 空間複雜度

依照不同的 CODING 方法以及 SET 中的數值排列順序，可分為兩種。

Best case：$O(\log^2 n)$

Worst case：$O(N^2)$