B10815057 Algorithms homework1

1

1.1 不是 max-heap

1.2 可以

1.3 因為只有資料 6 不符，與資料 7 交換(一次 MAX-HEAPIFY)後就符合

2

2.1 i > heapsize[A]/2，因為 leaf node 的數量是 heapsize[A]/2 取上高斯，所以 i 節點必定是 leaf node，而 leaf node 自身已是 max-heap，只需花費常數時間(判斷是否需要交換)就可完成 MAX-HEAPIFY，因此沒有影響。

3

3.1 任意刪除一個 heap 節點，該節點不一定比最後節點還大，若是用大於刪除節點的最後節點替換，則會造成 increase key，此時就必須使用 HEAP-INCREASE-KEY 來解決，若是一般狀況使用 MAX-HEAPIFY 即可解決，由於 HEAP-INCREASE-KEY 與 MAX-HEAPIFY，時間複雜度皆為 O(lgn)，故此演算法為 O(lgn)

A[i]-----> i th node in heap 要刪除的節點

A[A.heapsize]-----> last node in heap 最後節點

HEAP-DELETE (A, i)

If A[i] >= A[A.heapsize] //判斷刪除節點是否大於最後節點，如果大於，則替代後可能小於子節點，故需使用 MAX-HEAPIFY 來向下遞迴

A[i] = A[A.heapsize] //用最後一個節點取代要刪除的節點

A.heapsize = A.heapsize-1 //刪除最後一個節點，相當於 size-1

MAX-HEAPIFY(A,i) //取代後可能造成不符合 max-heap 規則，所以要 HEAPIFY

Else //若小於最後節點，執行 MAX-HEAPIFY 後可能大於父節點，故需使用 HEAP-INCREASE-KEY 來向上遞迴

HEAP-INCREASE-KEY (A,i, A[A.heapsize]) //取代後可能造成不符合 max-heap 規則，所以要 INCREASE-KEY

A.heapsize = A.heapsize-1 //刪除最後一個節點，相當於 size-1

4

4.1 最佳是 insertion sort，因為 insertion sort 是將某一個資料由後向前掃描，放入正確位置，所以就算有新資料加入也不需要整個 sorting 重做，但 selection 與 quick sort 若有新資料加入，若不處理或重做，會導致最終結果錯誤，因此 insertion sort 相對其他 2 個演算法不需要額外處理的時間，時間複雜度會保持相同。

5

5.1 這三個演算法都能找到最大的三筆資料

5.2 因為這三個演算法都能完整將 sequence 排序，然後取前三個元素就是最大的三筆資料

5.3

Insertion:O(n^2)從第二個元素到最後，每個元素會從後面往前掃過已排序的部分，一邊把資料往後搬，直到遇到比自己大的元素，就放在該元素之後，每個元素都插入過後，整個序列就是排序好的。

```
INSERTION-SORT(A)
    For i = 2 to A.length
        Target =A[i]
        j=i-1
        while j>0 and A[j] < Target
            A[j+1] = A[j]
            j=j-1
        A[j+1] = Target
```

Selection: O(n^2)一開始的處理範圍為整個序列，每次將處理範圍中最大的資料移到處理範圍的最前面，然後將處理範圍縮減一格，直到處理範圍為 0，因為每次都是找最大的，所以整個序列就會變成:第 1 大, 第 2 大, 第 3 大...第 n 大，這樣就是排序完成

```
SELECTION-SORT(A)
    For i = 0 to A.length
        For j = i+1 to A.length
            If A[i] < A[j]
                Swap(A[i],A[j])
```

Bubble: O(n^2)一開始的處理範圍為整個序列，每次比較相鄰的元素，會將越大的元素往前推，然後縮減處理範圍直到為 0，與 Selection sort 類似，最後就排序完成

```
BUBBLE-SORT(A)
    For i = 0 to A.length
        For j = 0 to A.length - i -1
            If A[j] < A[j+1]
                Swap(A[j], A[j+1])
```

6

6.1

Insertion:此演算法是插入到第一個大於等於元素的後一位置，因此相同大小的元素不會被改變順序

```
while j>0 and A[j] < Target //若是遇到 A[j] ==Target 的情形就會跳出迴圈，放到後一個位置
        A[j+1] = A[j]
        j=j-1
A[j+1] = Target
```

Merge:會將序列遞迴得區分成許多對數字，然後在合併時完成排序，雖然序列被區分為好幾個部分，但部分之間彼此的順序還是保留著，因此遇到相同元素時，只需依照部分之間的順序即可。

如下圖，若 L[i]==R[j]，則進入 then 處，取代 A[k]的就是 L[i]，而下次迴圈就會進入 else 處，所以 L[i]在左，R[j]在右，仍然保持正確順序

```
for k ← p to r
    do if L[ i ] ≤ R[ j ]
       then A[k] ← L[ i ]
            i ←i + 1
       else A[k] ← R[ j ]
            j ← j + 1
```

Selection:每次迴圈會將範圍內最大的值放到前面，但由於可能是透過非相鄰的元素交換，所以相同大小的元素在這個過程中，就可能被破壞順序，

舉例:序列 5 8 5 2 9 要由小排到大，第一遍選擇第 1 個元素 5 會和 2 交

換，此時的 2 個 5 順序就被調換了，因此 selection sort 是不穩定的

If A[i] < A[j] //如果 A[i]與後面的任一元素相同大小，交換後就會破壞順序

Swap(A[i],A[j])

Bubble:氣泡排序法的每次比較都是相鄰的元素在比，因此如果相同就不交換，即可不破壞順序

If A[j] < A[j+1] //如果相同就不會執行下面的交換

Swap(A[j], A[j+1])

Heapsort:因為創建 heap 的過程與第一次拿出根結點並將最後節點放到根結點後，順序就會被破壞了，舉例:

原始:{1, 5, 2a, 3, 2b, 6, 2c}

創建 heap 後(array representation): {1, 2b, 2a, 3, 5, 6, 2c}

排序後結果: {1, 2c, 2b, 2a, 3, 5, 6}

Quicksort:每次將某個元素放到最終正確的位置，但在過程中，可能會使用到多次交換，因此順序非常容易被破壞，如下圖黑色方框處

```
void partition(iterator begin, iterator end, iterator& loc, compare comp = compare()) {//把piviot放到對的位置(左邊都小於，右邊都大於)
        iterator left = begin, right = end;//起始點、終點
        loc = begin;//piviot設成起始點
        while (true) {
                while (comp(*loc,*right) && (loc != right)) {//從終點(right)往左比較，直到遇到比自己小的，就停下
                        --right;//往左走
                }
                if (loc == right) {//如果自己是最小的，代表piviot的位置已經是正確的，結束函式
                        return;
                }
                else {//如果不是最小的，就把piviot的位置換到新的位置，但還不是正確位置，繼續做
                        swap(*loc, *right);//交換
                        loc = right;//piviot指到新的位置
                }
                while (!comp(*loc, *left) && (loc != left)) {//從起點(left)往右比較，直到遇到比自己大的，就停下
                        ++left;//往右走
                }
                if (loc == left) {//如果自己是最大的，代表piviot的位置已經是正確的，結束函式
                        return;
                }
                else{//如果不是最大的，就把piviot的位置換到新的位置，但還不是正確位置，繼續做
                        swap(*loc, *left);//交換
                        loc = left;//piviot指到新的位置
                }
        }
}
```