Algorithm 2017 Spring
Homework 2 Solutions

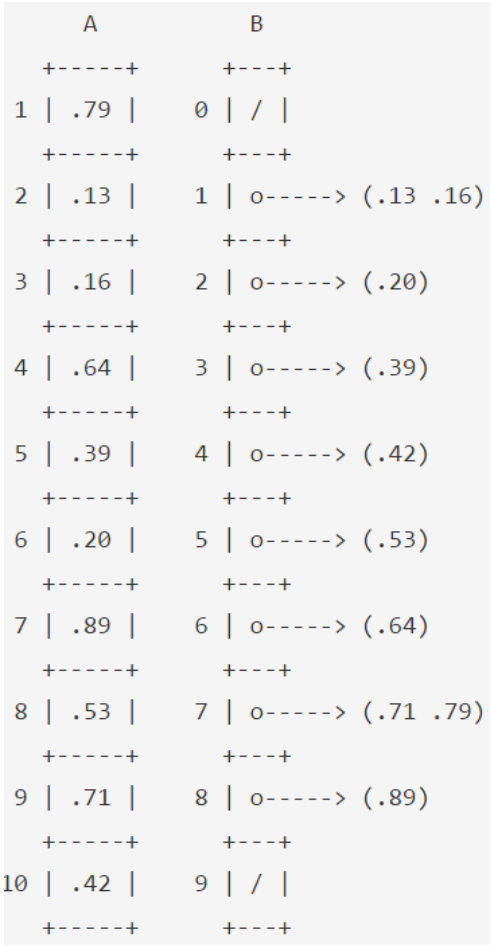
指導教授：謝孫源 教授

助教：許景添 陳琮皓 林玉陞 何岱璇

1. (10pts) Illustrate the operation of RADIX-SORT on the following list of English words: COW, DOG,, RUG, ROW, MOB,, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

COW		SEA		TAB		BAR
DOG	=>	TEA	=>	BAR	=>	BIG
SEA		MOB		EAR		BOX
RUG		TAB		TAR		COW
ROW		DOG		SEA		DIG
MOB		RUG		TEA		DOG
BOX		DIG		DIG		EAR
TAB		BIG		BIG		FOX
BAR		BAR		MOB		MOB
EAR		EAR		DOG		NOW
TAR		TAR		COW		ROW
DIG		COW		ROW		RUG
BIG		ROW		NOW		SEA
TEA		NOW		BOX		TAB
NOW	=>	BOX	=>	FOX	=>	TAR
FOX		FOX		RUG		TEA

2.(10pts) Illustrate the operation of BUCKET-SORT on the array A= ⟨.79,.13,.16,.64,.39,.20,.89,.53,.71,.42⟩.



3. What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order? What about decreasing order?

Let's assume that the heap is a full binary tree with $n = 2^k - 1$.

There are $2^k - 1$ leaves and $2^{k-1} - 1$ inner nodes.

Let's look at sorting the first $2^k - 1$ elements of the heap.

Let's consider their arrangement in the heap and color the leaves to be red and the inner nodes to be blue.

The colored nodes are a subtree of the heap (otherwise there would be a contradiction).

Since there are 2^{k-1}colored nodes, at most 2^{k-2} are red, which means that at least $2^{k-2} - 1$ are blue.

3. What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order? What about decreasing order?

While the red nodes can jump directly to the root, the blue nodes need to travel up before they get removed. Let's count the number of swaps to move the blue nodes to the root. The minimal case of swaps is when

- (1) there are $2^{k-2} - 1$ blue nodes
- (2) they are arranged in a binary tree.

If there are d such blue nodes, then there would be $i = \log d$ levels, each containing 2^i nodes with length i.

Thus the number of swaps is:

$$\sum_{i=0}^{\log d} i2^i = 2 + (\log d - 2)2^{\log d} = \Omega(d \log d)$$

3. What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order? What about decreasing order?

And now for a lazy (but cute) trick.

We've figured out a tight bound on sorting half of the heap. We have the following recurrence:

$$T(n) = T\left(\frac{n}{2}\right) + \Omega(n \log n)$$

Applying the master method, we get that$T(n) = \Omega(n \log n)$

3. What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order? What about decreasing order?

Both of them are $\Theta(n \lg n)$.

If the array is sorted in increasing order, the algorithm will need to convert it to a heep that will take $\mathcal{O}(n)$. Afterwards, however, there are $n-1$ calls to `MAX-HEAPIFY` and each one will perform the full $\lg k$ operations. Since:

$$\sum_{i=1}^{n-1} \lg k = \lg\Big((n-1)!\Big) = \Theta(n \lg n)$$

Same goes for decreasing order. `BUILD-MAX-HEAP` will be faster (by a constant factor), but the computation time will be dominated by the loop in `HEAPSORT`, which is $\Theta(n \lg n)$.

建立MaxHeap： O(n)
執行n-1次Delete Max：(n-1) × O(log n) = O(n log n)

答案各三分，說明各兩分

4. Prove that COUNTING-SORT is stable.

COUNTING-SORT is stable because the lines nine to eleven of the algorithm work in the following way:

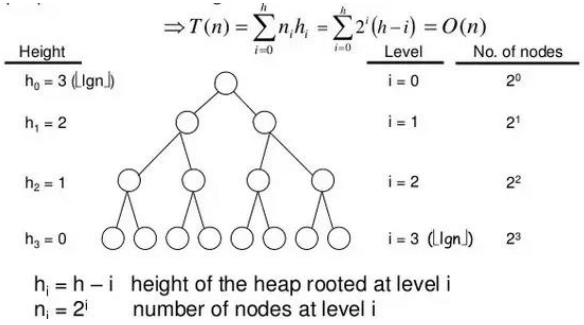
- It inserts the last occurrence in the originally unsorted array in the last possible position for that particular value.
- Future occurrences of that same value cannot be located but before the next occurrence of that value in the input array.

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

4. Prove that COUNTING-SORT is stable.

- Let's say that two elements at indices i1<i2 are equal to each other. In the sorted array, they take place at indices j1+1=j2. Since the COUNTING-SORT processes the input array in reverse order, A[i2] is put in B[j2] first and then A[i1] is put in A[j2]. Since the two elements preserve their order, the algorithm is stable.

因為此題是證明題，只有舉例不給分



5. In a heap:

A. (3pts) What are the minimum numbers of elements if the height is *h*? Show your solution process.

- Sol.
  - $\min = 2^0 + 2^1 + 2^2 + \cdots + 2^{h-1} + 1 = \frac{1-2^h}{1-2} + 1 = 2^h$

B. (3pts) What are the maximum numbers of elements if the height is *h*? Show your solution process.

- Sol.
  - $\min = 2^0 + 2^1 + 2^2 + \cdots + 2^h = \frac{1-2^{h+1}}{1-2} + 1 = 2^{h+1} - 1$

5. In a heap:

C. (4pts) Show that an n-element heap has height $\lfloor \log n \rfloor$.

- **Sol.**
  - The minimum number of elements is 2^h
  - The maximum number of elements is $2^{h+1} - 1$
  - So, $2^h \leq n \leq 2^{h+1} - 1 < 2^{h+1}$ by taking log
    - => $h \leq \log n < h + 1$ => $h = \lfloor log\, n \rfloor$.

6.(10pts) Show that the running time of QUICKSORT is (n^2), when the array A contains distinct elements and is sorted in decreasing order.

- In this case PARTITION always returns p because all the elements are greated than the pivot.
- While the if will never be executed, we still get one empty partition and the recurrence is $T(n) = T(n-1) + \Theta(n)$ (even if its body is not executed, the for is still Θ(n)).

- 法二：

- In each partition, the pivot element is always the smallest element. Therefore, each partition will produce two subarrays. The first subarray contains only one element (the smallerest element) and this element is in its correct position. The second subarray contains the remaining elements.
  In this case, the running time recurrence will be $T(n) = T(n-1) + n$. Thus, $T(n) = \Theta(n^2)$.

必須要寫出$T(n) = T(n-1) + \Theta(n)$ ，沒有只能得一半分數

7.

Setting up the initial values for the min and max depends on whether *n* is odd or even.

- If *n* is even, compare the first two elements and assign the larger to max and the smaller to min. Then process the rest of the elements in pairs.
- If *n* is odd, set both min and max to the first element. Then process the rest of the elements in pairs.

7.

If n is even, we do 1 initial comparison and then $3(n-2)/2$ more comparisons.

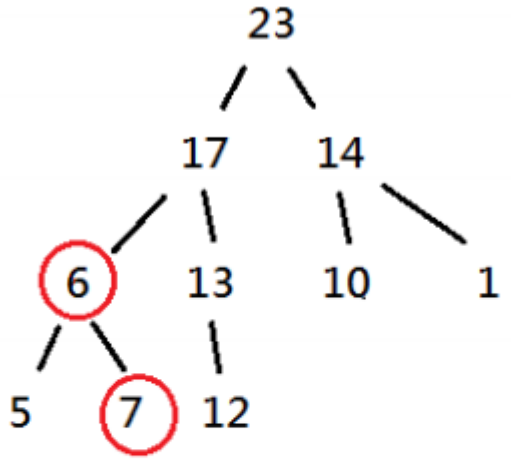
$$\begin{aligned}\#\text{ of comparisons} &= \frac{3(n-2)}{2}+1 \\ &= \frac{3n-6}{2}+1 \\ &= \frac{3n}{2}-3+1 \\ &= \frac{3n}{2}-2.\end{aligned}$$

If n is odd, we do $3(n-1)/2 = 3\lfloor n/2 \rfloor$ comparisons.

8.

解答

No, max-heap child值須小於parent



配分(10%)

扣分方式

只寫答案沒寫理由扣5分

9. Answer True of False for each the following statements on sorting algorithms.

(1) We can use a random number generator to improve the performance of Quick sort.

True

(2) Radix sort can only be performed on sequential lists, not on linked lists.

False

(3) The time complexity for comparison-based sorting algorithm is Ω(n log n).

True

(4) Radix sort requires the minimum data space.

False

(5) The conditions of worst case are the same for bubble sort and quick sort.

True

10.

You are told that a list of 10,000 words is already in order but you wish to check it to make sure and sort any words found out of order. Which of the following sorting algorithms would you choose: insertion sort, bubblesort, mergesort or quicksort? Explain your answer clearly.

用insertion sort, bubble sort。

∵ 此10,000 words 已經大部份均sort 過，只是要確定是否有out of order。

∴ 用insertion sort or bubble sort 可在其確定sort 好之後立即結束，較其它algorithm 有效率。