Algorithm Hw2 Solution

指導教授：謝孫源 教授

助教: 葉承翰、詹博丞、林帝丞

Question 1

解答:

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

Question 2

解答:

- **Initialization.**
  The subarray A[i+1…n] is empty, thus the invariant holds.
- **Maintenance.**
  A[1] is the largest element in A[1…i] and it is smaller than the elements in A[i+1…n]. When we put it in the ith position, then A[i…n] contains the largest elements, sorted. Decreasing the heap size and calling **MAX-HEAPIFY** turns A[1…i−1] into a max-heap.
  Decrementing i sets up the invariant for the next iteration.
- **Termination.**
  After the loop i=1. This means that A[2…n] is sorted and A[1] is the smallest element in the array, which makes the array sorted.

Question 3

解答:

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.

Question 3

解答:

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 and

(2) they are arranged in a binary tree.

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

Thus the number of swaps is:

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

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:

$$T(n) = T(n/2) + \Omega(n \lg n)$$

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

Question 4

解答:

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)).

Question 5

解答:

In the worst case, the number of calls to RANDOM is:

$$T(n) = T(n-1) + 1 = n = \Theta(n)$$

As for the best case:

$$T(n) = 2T(n/2) + 1 = \Theta(n)$$

This is not too surprising,

because each third element (at least) gets picked as pivot.

Question 6

解答:

The best case happens when the partition is even, that is:

$$T(n) = 2T(n/2) + \Theta(n)$$

Using the master method, we get the solution

$$\Theta(n \lg n).$$

Question 7

解答:

A:

6	0	2	0	1	3	4	6	1	3	2

We build an array of counts:

C:

0	1	2	3	4	5	6
2	2	2	2	1	0	2

The number of elements before each

C:

0	1	2	3	4	5	6
2	4	6	8	9	9	11

Then we start iterating

Question 7

解答:

Then we start iterating

B:

1	2	3	4	5	6	7	8	9	10	11
					2					

C:

0	1	2	3	4	5	6
2	4	5	8	9	9	11

B:

1	2	3	4	5	6	7	8	9	10	11
					2		3			

C:

0	1	2	3	4	5	6
2	4	5	7	9	9	11

B:

1	2	3	4	5	6	7	8	9	10	11
			1		2		3			

C:

0	1	2	3	4	5	6
2	3	5	7	9	9	11

B:

1	2	3	4	5	6	7	8	9	10	11
			1		2		3			6

C:

0	1	2	3	4	5	6
2	3	5	7	9	9	10

B:

1	2	3	4	5	6	7	8	9	10	11
			1		2		3	4		6

C:

0	1	2	3	4	5	6
2	3	5	7	8	9	10

Question 7

解答:

Then we start iterating

B:

1	2	3	4	5	6	7	8	9	10	11
			1		2	3	3	4		6

C:

0	1	2	3	4	5	6
2	3	5	6	8	9	10

B:

1	2	3	4	5	6	7	8	9	10	11
		1	1		2	3	3	4		6

C:

0	1	2	3	4	5	6
2	2	5	6	8	9	10

B:

1	2	3	4	5	6	7	8	9	10	11
	0	1	1		2	3	3	4		6

C:

0	1	2	3	4	5	6
1	2	5	6	8	9	10

B:

1	2	3	4	5	6	7	8	9	10	11
	0	1	1	2	2	3	3	4		6

C:

0	1	2	3	4	5	6
1	2	4	6	8	9	10

B:

1	2	3	4	5	6	7	8	9	10	11
0	0	1	1	2	2	3	3	4		6

C:

0	1	2	3	4	5	6
0	2	4	6	8	9	10

B:

1	2	3	4	5	6	7	8	9	10	11
0	0	1	1	2	2	3	3	4	6	6

C:

0	1	2	3	4	5	6
2	3	5	7	8	9	9

Question 8

解答:

Let's say that two elements at indices $i_1 < i_2$ are equal to each other.

In the sorted array, they take place at indices $j_1 + 1 = i_2$.

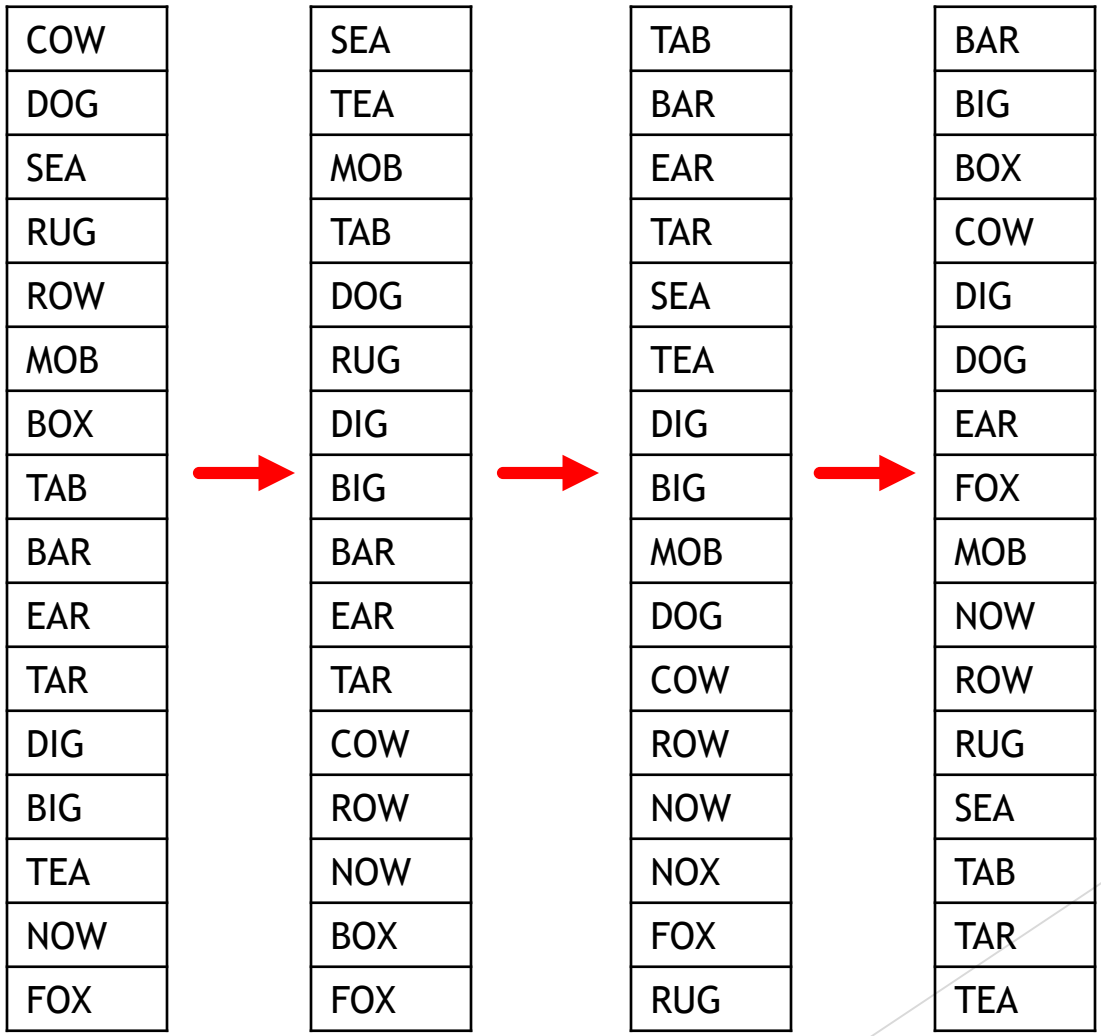
Since the COUNTING-SORT processes the input array in reverse order,

A[i_2] is put in B[j_2] first and then A[i_1] is put in A[j_2].

Since the two elements preserve their order, the algorithm is stable.

Question 9

解答:



Question 10

解答:

A	
1	0.79
2	0.13
3	0.16
4	0.64
5	0.39
6	0.20
7	0.89
8	0.53
9	0.71
10	0.42

B		
0	/	
1	->	0.13-> 0.16
2	->	0.20
3	->	0.39
4	->	0.42
5	->	0.53
6	->	0.64
7	->	0.71-> 0.79->
8	->	0.89->
9	/	