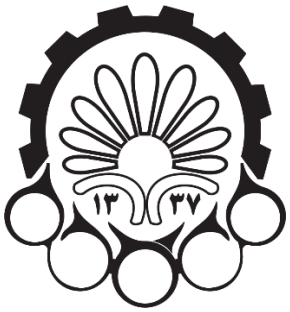
بسمه تعالی



دانشگاه صنعتی امیرکبیر **دانشکده مهندسی کامپیوتر**

پاسخنامه تمرین سری دوم درس طراحی الگوریتم‌ها

1. Consider calling HEAPSORT on an array which is sorted in decreasing order. Every time $A[1]$ is swapped with $A[i]$, MAX-HEAPIFY will be recursively called a number of times equal to the height h of the max-heap containing the elements of positions 1 through $i-1$, and has runtime $O(h)$. Since there are 2^k nodes at height k, the runtime is bounded below by

$$\sum_{i=1}^{\lfloor \lg n \rfloor} 2^i \log(2^i) = \sum_{i=1}^{\lfloor \lg n \rfloor} i2^i = 2 + (\lfloor \lg n \rfloor - 1)2^{\lfloor \lg n \rfloor} = \Omega(n \lg n).$$

2. Since the call on line one could possibly take only linear time (if the input was already a max-heap for example), we will focus on showing that the for loop takes n log n time. This is the case because each time that the last element is placed at the beginning to replace the max element being removed, it has to go through every layer, because it was already very small since it was at the bottom level of the heap before.

3. Construct a min heap from the heads of each of the k lists. Then, to find the next element in the sorted array, extract the minimum element (in $O \lg(k)$ time). Then, add to the heap the next element from the shorter list from which the extracted element originally came (also $O(\lg(k))$ time). Since finding the next element in the sorted list takes only at most $O(\lg(k))$ time, to find the whole list, you need $O(n \lg(k))$ total steps.

4. The running time of QUICKSORT on an array in which evey element has the same value is n^2. This is because the partition will always occur at the last position of the array so the algorithm exhibits worst-case behavior.

5. If the array is already sorted in decreasing order, then, the pivot element is less than all the other elements. The partition step takes $\Theta(n)$ time, and then leaves you with a subproblem of size $n-1$ and a subproblem of size 0. This gives us the recurrence $T(n) = T(n-1) + \Theta(n)$. Then we use the substitution method to to prove that this recurrence has the solution $T(n) = \Theta(n^2)$:

By definition of Θ, we know that there exists c_1, c_2 so that the $\Theta(n)$ term is between $c_1 n$ and $c_2 n$. We make that inductive hypothesis be that $c_1 m^2 \leq T(m) \leq c_2 m^2$ for all $m < n$, then, for large enough n,

$$c_1 n^2 \leq c_1(n-1)^2 + c_1 n \leq T(n-1) + \Theta(n)$$

$$= T(n) = T(n-1) + \Theta(n) \leq c_2(n-1)^2 + c_2 n \leq c_2 n^2$$

6. We can construct the graph whose vertex set is the indices, and we place an edge between any two indices that are compared on the shortest path. We need this graph to be connected, because otherwise we could run the algorithm twice, once with everything in one component less than the other component, and a second time with the everything in the second component larger. As long as we maintain the same relative ordering of the elements in each component, the algorithm will take exactly the same path, and so produce the same result. This means that there will be no difference in the output, even though there should be. For a graph on n vertices, it is a well known that at least $n-1$ edges are necessary for it to be connected, as the addition of an edge can reduce the number of connected components by at least one, and the graph with no edges has n connected components.

So, it will have depth at least $n-1$.

7. First run through the list of integers and convert each one to base n, then radix sort them. Each number will have at most $\log_n(n^3) = 3$ digits so there will only need to be 3 passes. For each pass, there are n possible values which can be taken on, so we can use counting sort to sort each digit in $O(n)$ time.

8. In the worst case, we could have a bucket which contains all n values of the array. Since insertion sort has worst case running time $O(n^2)$, so does Bucket sort. We can avoid this by using merge sort to sort each bucket instead, which has worst case running time $O(n \lg n)$.