CS2351 Data Structures

Homework 2 (Suggested Solution)

There are many different ways to answer the questions perfectly. Don't worry if your answer is different with ours. In general, what we want to have is *precise and concise.*

1. **Ans:**

   (a) We guess $T(n) = O(n)$ and show it by induction. Assume that $T(n) \leq cn$ for all $n < k$. Then we have:

   $$\begin{aligned} T(k) &= T(k/5) + T(3k/4) + k \leq ck/5 + 3ck/4 + k \\ &= 19ck/20 + k \leq ck \quad \text{whenever } c \geq 20. \end{aligned}$$

   Thus, $T(n) \leq 20n$ for all n. On the other hand, by definition of $T(n)$, we have $T(n) \geq n$, so that $T(n) = \Omega(n)$. Combining the results, we conclude that $T(n) = \Theta(n)$.

   (b) Since $n^{\log_2 8} = n^3 = \Theta(n^3)$, by Master Theorem, we have

   $$T(n) = \Theta(n^3 \log n).$$

   (c) Let $m = \log n$ and $S(m) = T(2^m) = T(n)$. Then we have:

   $$S(m) = 8S(m/2) + m^3.$$

   From the result of part (b), we see that $S(m) = \Theta(m^3 \log m)$. Thus, we have:

   $$T(n) = S(m) = \Theta(\log^3 n \log \log n).$$

2. **Ans:**

   (a) Quick Sort sorts 1 element correctly. Inductively, assume that Quick Sort sorts n elements correctly, whenever $n < k$. When Quick Sort is applied on an array of k elements, it first selects an arbitrary element x, and obtain the groups A_{small} and A_{large}. Since both groups must contain fewer than k elements, Quick Sort must sort each group correctly (due to the induction hypothesis). The overall correctness follows since all elements in A_{small} are smaller than x, and all elements in A_{large} are larger than x, so that the returned array of k elements is sorted.

   (b) If in each round, the element x selected partition the groups unevenly, say all elements are in A_{large}, the total running time will be

   $$(n-1) + (n-2) + \cdots + 1 = \Theta(n^2).$$

   Since the worst case can only be worse, the worst case running time is $\Omega(n^2)$.

   (c) The algorithm may not return a sorted array with the same number of elements. For instance, if all elements are the same, there will be no A_{small} and A_{large}, and the algorithm will just return 1 element!

   (d) To make the algorithm correct, there are two methods. One is extend A_{small} to contain those elements with the same value as x. Another method is to count the number t of elements with value equal to x, so that in the final output, we first list the sorted A_{small}, then list t copies of x, and then list the sorted A_{large}. The second method is slightly better since there are fewer level of recursions.

3. We first insert the n numbers into the heap, but without caring the heap property. Next, we use the heapify process to fix the heap property of all nodes. Finally we call Extract-Min k times to obtain the desired k smallest numbers, in sorted order.

   The time for heapify is $O(n)$ and the total time for k Extract-Min is $O(k \log n)$. Thus the above algorithm runs in $O(n + k \log n)$ time.