ESO 207

Hints and Solutions: Practice Problems - 2

Devendra Agrawal and Rachit Nimavat
TA, ESO207

September 6, 2014

1. For each half you need to store 4 things, Best Sum of the Half , Best Prefix Sum and Best Suffix Sum of the Half and the overall sum of the Half. Now using data from both the halves , we can easily calculate the data for merged portion. Let us call both half's as H1 and H2 and we are interested in merging them to H, then this will hold:
   $H.Sum = H1.sum + H2.sum$
   $H.BestPrefix = max(H1.BestPrefix, H1.Sum + H2.BestPrefix)$
   $H.BestSuffix = max(H2.BestSuffix, H2.Sum + H1.BestSuffix)$
   $H.BestSum = max(H1.BestSum, H2.BestSum, H1.BestSuffix+H2.BestPrefix)$

   $\mathcal{T}(N) = 2 * \mathcal{T}(N/2) + \mathcal{O}(1)$.

2. Main Idea for the problem is that if two cards are distinct then even if you remove both of them then also the card which had more than $n/2$ copies of itself, would have more then $(n-2)/2$ copie of itself in the remaining set of $n-2$ cards. At the end of the algorithm you will get a collection of cards which are same cards, you can detect whether this card is present more than $n/2$ times in 1 more pass.

   $\mathcal{T}(N) = 2 * \mathcal{T}(N/2) + \mathcal{O}(N)$.
   Can you modify it to get $\mathcal{O}(N)$ time algorithm?

3. Basically, one can extend Merge-sort. Recursively, sort the left half, and sort the right half, and for each of the halves, count and add the number of significant

inversions.
Now we can merge the two halves to maintain the invariant. Also, in the merge phase, the two halves are sorted, so we can count the heavy inversions, in which the heavier element is on the left half and the lighter is on the right half. This can be done something like this.
Start with the heaviest element of the left half, $i = n/2$. Find the heaviest element with index say j in the right half such that i, j is a significant inversion. This contributes 1x$j = j$ significant inversions. Now shift i to the left by 1, and shift j to the left until we find the heaviest element in index j that corresponds to a significant inversion. This contributes j , which we add to the running sum. Overall we make a pass over the left half and a pass over the right half, so this takes $\mathcal{O}(N)$ time. Overall the recurrence is $\mathcal{T}(N) = 2 * \mathcal{T}(N/2) + \mathcal{O}(N)$ giving $\mathcal{O}(Nlog\ N)$ bound.

4. Main Idea for the problem is to exploit the property of the heap that the child will have more value than the child. So if the root of a subtree has value greater than x , then every child will also have value greater than x ,and it will be of no use to traverse their children. So you start from the root, and you will stop when you get k values which are smaller than x. So, Overall time complexity would be $\mathcal{O}(k)$.

5. Suppose there are k such intervals given. Let L be the maximum element of any of the intervals. Then you can find n^{th} smallest element in $\mathcal{O}(klog\ L)$ time by doing a binary search from 0 to L. Suppose, at any instant, you need to find the rank of element x. You can do that in $\mathcal{O}(k)$ time, which results in the above algorithm.