ESO 207

Hints and Solutions: Practice Problems - 1

Devendra Agrawal and Rachit Nimavat
TA, ESO207

September 6, 2014

1. **Idea :** Find the sum of all numbers in the array , let's call it as Sum_from_Array. Find the sum of all numbers from 1 to N, let's call it as Total_Sum. The remaining number will be Total_Sum - Sum_from_Array , proof is left as exercise.
   a. Find the sum and the sum of squares of the numbers, let missing numbers be A and B , then $A + B = (1 + 2... + N)-$ Sum_from_Array and $A^2 + B^2 = (1^2 + 2^2... + N^2)-$Sum_of_Squares_from_Array, using these two equations , you can find the solution for the missing number.
   b. In this part ,we really need to do bucket sort , but we are not given extra space,the only challenge is to do bucket sort in the same array itself. So let A be the array of size $n - k$, then add k elements to A each with INF. Main idea is to get $A[i] = i$, for as many i as possible. Read the psedo code to get the idea:

   for i in (1,n): while (A[A[i]] != A[i]) swap (A[A[i]] , A[i]);

   for i in 1 to n **do**
       while $A[A[i]]! = A[i]$ **do** swap($A[A[i]]$, $A[i]$)
       end while
   end for

   Finally , output the index which are INF.
   If you have understood the solution nicely , then try solving :
   There is an array of N numbers , each number is from 1 to N , except exactly one number which has occured twice (and exactly one number which is missing) , find that number in O(N) complexity and O(1) space.

2. This is very simple one. Help yourself out in this problem.

3. Make a Max heap of first k elements in $\mathcal{O}(klog\ k)$ time,then you can follow this:
   for i in $k+1$ to n **do**
   Extract-Max from Heap
   Report Max
   Insert A[i]
   end for

   Time Needed : $\mathcal{O}(log\ k)$ per loop , so overall $\mathcal{O}(nlog\ k)$ complexity is acheived.

4. This is a suproblem of first programming assignement.

5. Make a Max-Heap and pass over the array. Insert an element if the size of heap is less than K. Pop and insert an element if the new element has value less than the max element of the heap. Since the size of heap is never greater than k, each of the operation costs $\mathcal{O}(log\ k)$ resulting in $\mathcal{O}(nlog\ k)$ algorithm.

6. Straight Forward , try it out on paper, you can get the algorithm for sure :)

7. Make an $2D$ visited array to keep track of the positions you have visited. You start at some (a,b) and you are aiming to reach (c,d) in minimum steps. So when you start at (a,b) you already have visited (a,b). You also need to make distance array to keep track of the minimum distance from $(a.b)$ to (i,j). Then the algorithm is defined below.
   $Q \leftarrow \{(a,b)\}$
   $visited[(a,b)] \leftarrow True$
   $distance[(a,b)] \leftarrow 0$

   while Q is not empty **do**
   $Top \leftarrow Q.pop()$
   $L \leftarrow$ All unvisited positions from Top which can be reached by k in single step

   for each e in L **do**
   $visited[m] \leftarrow True$
   $distance[m] \leftarrow distance[Top]+1$
   $Q.push(m)$

   end for
   end while

   See that no elements are inserted in the queue more than once. How can you use this observation to find the time complexity?