COL 702 Advanced Data Structures and Algorithms

Minor 1, Sem I 2017-18, Max 40, Time 1 hr

Name ______________________ **Entry No.** ______________________ **Group**

Note Write in the space provided below the question including back of the page.
Every algorithm must be accompanied by a proof of correctness, time and space complexity. You can however quote any result covered in the lectures without proof.

1. To sort n numbers in minimum time, in the range $[1, m]$, for what value of m should we use (express m as a function of n) ? **(5 marks)**
   (i) Heapsort (ii) Radix sort

   m requires $\log m$ bits to be represented,so it will take $O(n\frac{\log m}{\log n})$ using radix sort and $O(n \log n)$ comparisons using heapsort. For $\log m = \omega(\log^2 n)$ which is $m \geq n^{\log n}$, we should use heapsort and radix sort otherwise.

   Note that if comparison time scale up as number of bits, then the time for heapsort becomes $O(n \log n \log m)$ which is always larger than radix sort.

2. Given a binary (min) heap $\mathcal{H}$, describe an algorithm to output all elements in $\mathcal{H}$ that have values less than a given number x. Analyse the performance of your algorithm in terms of the number of elements output, m and the size of the heap denoted by n.

   We do a kind of depth first search using x. If x is smaller than the root, we don't search in any of its subtrees or else we output the root and recursive search the subtrees.

   The time spent is the size of output plus the number of times, we do not explore the two subtrees. This is $\leq 3m$ since a node has at most 2 children. Note that it is independent of n. **(5 marks)**

3. In a given unordered sequence S of numbers $x_1, x_2, \ldots x_n$, a pair (x_i, x_j) is *inverted* if $j > i$ and $x_i > x_j$. Design an efficient algorithm to count the number of inversions in a given sequence of length n. For example,in the sequence $[3.5, 7, 6, 2.1]$, the number of inversions is 4 - corresponding to (3.5, 2.1) , (7,6) (7, 2.1) , (6, 2.1).

   Your algorithm should be faster than the brute force $O(n^2)$ approach that inspects every pair. **(1+9 marks)**
   (i) What is the maximum number of inversions in an n element sequence and which permutation corresponds to it ?

   For a descending sequence this is $\binom{n}{2}$, i.e., all pairs. (ii) Algorithm for inversion with analysis

   The mergesort algorithm can be easily extended to compute the number of inversions. It is a divide and conquer approach. Suppose we have counted the number of inversions in the first half of the array and in the second half of the array recursively. We have also sorted the first and second halves recursively. Then, the additional inversions between elements of the first half and the second half can be computed as follows. Any element $x \in H_1$ (first half) is inverted with an element $y \in H_2$ (second half) if $y < x$. In other words, the rank of x in H_2 is the number of innversions involving x. This is easily computed at the time of merging by keeping track of how many elements from H_2 were output before x. The analysis is based on the recurrence

   $$T(n) = 2T(n/2) + O(n) \quad T(2) = 1 \quad \Rightarrow T(n) = O(n \log n)$$

4. You are given an undirected weighted graph $G = (V, E)$ (without negative weights) that has some of the vertices $W \subset V$ designated as petrol pumps. Imagine that the graph represents the road network of a city and we want to find the closest petrol-pump from any location (vertex). Design an efficient

algorithm for this problem, which should output for every vertex $v \in V$, its closest petrol-pump $w \in W$. For k petrol pumps ($k \leq n$), your algorithm should be faster than $O(kn)$, i.e., running Dijkstra's algorithm repeatedly. Here $n = |V|, m = |E|$. **(10 marks)**

Create a new vertex, say r and add edges (r, v) for all $v \in W$ having 0 weights. Run Dijkstra's SSSP algorithm from r. The distances to the vertices $v \in V$ is the distance from v to the nearest vertex $w \in W$.

Proof (of correctness) All the shortest paths from r go through W, and all these edges have 0 weight. So the distance computed from r must go through the vertex in W that is closest to any vertex $v \in V - W$. Note that for $v \in V$ the distance must be 0. You can also think in terms of all vertices in $w \in W$ have labels $\delta(w) = 0 = D(w)$ initially when we start running Dijkstra.

The running time is $O((m+n)\log n)$ since number of vertices added is 1 and number of edges added is $k \leq n$.

5. We build a binary search tree on n numbers using the following algorithm

   1. Choose a random element as root ∇ and divide the other elements into two sets $S_<$: elements smaller than the root and $S_>$: elements larger than the root.
   2. If any of sets is larger than $2n/3$, go to step 1.
   3. Build the left subtree on $S_<$ and right subtree on $S_>$ recursively, if they are non-empty.
   4. The root of $S_<$ ($S_>$) is the left (right) child of ∇.

   (3+7 marks)
   (i) What is the worst case time for searching for an element in the constructed tree ?

   Since every time, we go to the next level of recursion, the subproblems decrease by a factor 2/3 , the search interval also decreases by the same factor, so the height of the tree is at most $\log_{3/2} n = O(\log n)$. Note that this is *worst case* since the tree that is output by the algorithm is guaranteed to have the property. It is only the construction time that is probabilistic. (ii) What is the expected time to build this tree ?

   This is very similar to the analysis for selection except that we have to consider the time for both the subtrees. In level i , we may have 2^i subproblems $T_{i,1}, T_{i,2} \ldots T_{i,j} \ldots$ but the sum of the subproblems sizes is always bound by n, i.e. $\sum_i n_{i,j} \leq n$ where $n_{i,j}$ represents the size of the j-th subproblem in level i.

   The expected time for a subproblem $E[T_{i,j}] = 3n_{i,j}$ since the probability that we succeed in going to step 3 is 1/3 (so expected number of repeatitions is 3). Since the total running time $T = \sum_i \sum_j T_{i,j}$

$$E[T] = E[\sum_i \sum_j T_{i,j}] = \sum_i \sum_j E[T_{i,j}] = \sum_i \sum_j 3n_{i,j} \leq \sum_i 3n \leq 3n \log_{3/2} n$$

   Therefore expected running time for building this tree is $O(n \log n)$. Note that the inorder traversal of this tree gives the sorted sequence and that is the analysis of quicksort.