CS5050 Advanced Algorithms

Fall Semester, 2015

Assignment 4: Data Structures

Due Date: 8:30 a.m., Friday, Oct. 30, 2015 (**at the beginning of CS5050 class**)

1. Given k sorted lists $L_1, L_2, \ldots, L_k$, with $1 \leq k \leq n$, such that each list contains some numbers in sorted order (e.g., in descending order) and the total number of numbers in all lists is n, using a **heap**, design an $O(n \log k)$ (not $O(n \log n)$) time algorithm for sorting all n numbers in the k sorted lists. **(20 points)**

   Remark. An $O(n \log k)$ time algorithm would be better than an ordinary $O(n \log n)$ time sorting algorithm when k is much smaller than n (e.g., $k = O(\log n)$).

2. Let A be an array of n distinct elements that store a max heap (a max heap is one that stores the largest key at its root). Each element of A is also called a "key" of the heap. The largest key of A can clearly be reported in constant time (simply by looking at the key stored at $A[1]$, without removing it from A). We denote the operation of reporting the largest key in A (without removing that key from A) by *Report-Max*(1). A generalized operation of *Report-Max*(1), denoted by *Report-Max*(k) can be defined as follows: Given an integer k as input, with $1 \leq k \leq n$, report the largest k keys (i.e., the 1st, 2nd, ..., kth largest keys) in A without removing them from A. Design an algorithm to implement *Report-Max*(k) in $O(k \log k)$ time. (Note: Although k can be equal to n in the worst case, k in general is much smaller than n). **(20 points)**

   Remark. Such an $O(k \log k)$ time algorithm is an "output-sensitive" algorithm because the running time of the algorithm is a function of the output size.

3. This problem is concerned with **range queries** (we have discussed a similar problem in class) on a binary search tree T whose keys are real numbers (you may assume no two keys in T are the same). Let h denote the height of T. The range query is generalization of the ordinary *search* operation. The **range** of a range query on T is defined by a pair $[x_l, r_r]$, where x_l and x_r are real numbers and $x_l \leq x_r$. Note that x_l and x_r need not be keys stored in T.

   You already know that the binary search tree T can support the ordinary *search, insert*, and *delete* operations, each in $O(h)$ time. You are asked to give an algorithm to efficiently perform the *range queries*. That is, in each range query, we are given a range $[x_l, r_r]$, and your algorithm should report all keys x stored in T such that $x_l \leq x \leq x_r$. Your algorithm should run in $O(h + k)$ time, where k is the number of keys of T in the range $[x_l, x_r]$. In addition, it is required that all keys in $[x_l, x_r]$ be reported in a *sorted order*. **(20 points)**

4. Consider one more operation on the above binary search tree T in Question 3: *range-sum*(x_l, x_r). Given any range $[x_l, x_r]$ with $x_l \leq x_r$, the operation *range-sum*(x_l, x_r) reports the *sum* of the keys in T that are in the range $[x_l, x_r]$.

   You are asked to augment the binary search tree T, such that the *range-sum*(x_l, x_r) operations, as well as the normal *search, insert*, and *delete* operations, all take $O(h)$ time each, where h is the height of T.

   You must present: (1) the design of your data structure (i.e., how you augment T); (2) the algorithm for implementing the *range-sum*(x_l, x_r) operation. **(20 points)**

Total Points: 80