CS 457, Data Structures and Algorithms I
Third Problem Set

October 17, 2019

Due on October 25. Collaboration is not allowed. Contact Daniel and me for questions.

1. (11 pts) Prove tight **worst-case** asymptotic upper bounds for the following recurrence equation that depends on a variable $q \in [0, n/4]$. Note that you need to prove an upper bound that is true for every value of $q \in [0, n/4]$ and a matching lower bound for a specific value of $q \in [0, n/4]$ of your choosing. Do not assume that a specific q yields the worst case input; instead, formally identify the q which maximizes the running time. (Hint: look at the bottom of Page 180 for the analysis of the worst-case running time of Quicksort)

$$T(n) = \begin{cases} 1 & \text{if } n \leq 2 \\ T(n-2q-1) + T(3q/2) + T(q/2) + \Theta(1) & \text{otherwise.} \end{cases}$$

2. (24 pts) Given an array S of n distinct numbers provide $O(n)$-time algorithms for the following:
   - Given two integers $k, \ell \in \{1, \ldots, n\}$ such that $k \leq \ell$, find all the ith order statistics of S for *every* $i \in \{k, \ldots, \ell\}$.
   - Given some integer $k \in \{1, \ldots, n\}$, find the k numbers in S whose *values* are closest to that of the median of S.

3. (25 pts) Consider the following silly randomized variant of binary search. You are given a sorted array A of n integers and the integer v that you are searching for is chosen uniformly at random from A. Then, instead of comparing v to the value in the middle of the array, the randomized binary search variant chooses a random number r from 1 to n and it compares v with $A[r]$. Depending on whether v is larger or smaller, this process is repeated recursively on the left sub-array or the right sub-array, until the location of v is found. Prove a tight bound on the expected running time of this algorithm.

4. (20 pts) You are given a set S of n integers, as well as one more integer v.
   a) Design an algorithm that determines whether or not there exist two distinct elements $x, y \in S$ such that $x + y = v$. Your algorithm should run in time $O(n \log n)$, and it should return (x, y) if such elements exist and (NIL, NIL) otherwise.
   b) Formally explain why your algorithm runs in $O(n \log n)$ time.

5. (20 pts) Suppose that you are given a sorted array A of *distinct* integers $\{a_1, a_2, \ldots, a_n\}$, drawn from 1 to m, where $m > n$.
   a) Give an $O(\log n)$ algorithm to find an integer from $[1, m]$ that is not present in A. For full credit, find the smallest such integer.
   b) Formally explain why your algorithm runs in $O(\log n)$ time.