EC330 Applied Algorithms and Data Structures for Engineers
Fall 2018

Homework 5

Out: October 22, 2018
Due: November 5, 2018

This homework has a written part and a programming part. Both are due at 8:59 am on November 5. You should submit both parts on Blackboard. For the written part, you should submit a single PDF file containing either typeset answers or scanned copies of hand-written answers. Make sure you write your answers clearly. For the programming part, your code should be easy to read and understand, and demonstrate good code design and style. Your program must compile and run on the lab computers.

1. **Heap [10 pt]**
   Given eight numbers $\{n_1, n_2, \ldots, n_8\}$, show that you can construct a heap (either min-heap or max-heap) using eight comparisons between these numbers (e.g., comparing n_1 and n_2 would be one comparison).

2. **Median [10 pt]**
   Describe an efficient algorithm for finding the median of all numbers between *two sorted arrays each of size* n. Your algorithm should be asymptotically faster than $O(n)$. Analyze the running time of your algorithm.

3. **Hashing [10 pt]**
   Given a sequence of inputs 531, 43, 63, 79, 4, 99, 89 and the hash function h(x) = x mod 10, show the resulting hash table for each of the following cases of conflict resolution.
   (a) Chaining
   (b) Linear probing
   (c) Quadratic probing
   (d) Using a second hash function $h_2(x) = 7 - (x \bmod 7)$

4. **Programming [70 pt]**
   Make sure to write your name and BU ID in a comment at the top of the program, along with your collaborator's name and BU ID, if any. To use the compiler on the lab computers, run "*module load gcc*" first. **Submit only *prob4.h*.**
   a) **[35 pt]**
      Implement the function *findMedian* in *prob4.h*. Similar to Question 2, the function should return the median of all numbers between two sorted vectors *v1* and *v2*. However, in this case, the sizes of *v1* and *v2* may differ. Below some example input and output.

Input: v1 = [1, 2], v2 = [3]
Output: 2

Input: v1 = [1], v2 = [2]
Output: 1.5

Your algorithm should have an asymptotic running time faster than $O(n + m)$ where n and m are the sizes of the two vectors.

Test your code thoroughly and document your code clearly.

b) **[35 pt]**
Implement the function *maxCollinearPoints* in *prob4.h*. Given a set of points, it should find the maximum number of points that lie on the same line and return this set of points. If there is more than one set, return one of them. You may assume that all the input points are unique and there are at least two points in the input set. You may find it useful to define lines with slopes and y-intercepts. Below is an example input and output.

Input: {(1,1), (1,4), (2,3), (3,2)}
Output: {(1,4), (2,3), (3,2)}

Test your code thoroughly and document your code clearly. The most time/space efficient solution(s) will receive 5 bonus points.