University of Waterloo
CS240E, Winter 2022
Assignment 3

Due Date: Wednesday, February 16, 2022 at 5pm

Be sure to read the assignment guidelines (`http://www.student.cs.uwaterloo.ca/~cs240e/w22/guidelines/guidelines.pdf`). Submit your solutions electronically as individual PDF files named a3q1.pdf, a3q2.pdf, ... (one per question).

Question 0 Academic Integrity Declaration

Read, sign and submit A03-AID.txt now or as soon as possible.

Question 1 [6+6=12 marks]

Consider the following algorithm to find the minimum in a binary search tree.

Algorithm 1: *findMin*(root r)

if *(r is NIL)* **then return** *"empty tree"*
while $r.leftChild \mathrel{!=} NIL$ **do** $r \leftarrow r.leftChild$
return $r.key$

Let $T^{\text{avg}}(n)$ (for $n \geq 0$) be the average-case number of executions of the while-loop in *findMin* for a tree with n nodes. Here the average is taken over all binary search trees that store $\{0, \ldots, n{-}1\}$, and $T^{\text{avg}}(0) = T^{\text{avg}}(1) = 0$.

a) Show that for $n \geq 2$ we have $T^{\text{avg}}(n) \leq 1 + \frac{1}{C(n)} \sum_{i=0}^{n-1} C(n{-}i{-}1)C(i)T^{\text{avg}}(i)$, where $C(n)$ is the number of binary search trees that stores $\{0, \ldots, n{-}1\}$. Be as precise as we were in class for *avgCaseDemo*.

b) Show that $T^{\text{avg}}(n) \in O(\log n)$. (We recommend that you show $T^{\text{avg}}(n) \leq 2 \log n$, and that you consider a 'good case' where the left subtree has size at most $n/2$.) You may use without proof that $C(n) = \sum_{i=0}^{n-1} C(i) \cdot C(n{-}i{-}1)$, and you may assume that n is divisible as needed.)

Question 2 [5 marks]

Let S be a skip list with $n \geq 4$ items. Assume that the lists $S_0, S_1, \ldots, S_h$ of S have the following property for all $0 \leq i < h$.

$$\text{If } |S_i| = 1 \text{ then } |S_{i+1}| = 0. \text{ If } |S_i| > 1, \text{ then } |S_{i+1}| \leq \sqrt{|S_i|}.$$

What is the maximum possible value of h, relative to n? For full marks, you should give an exact bound (no asymptotics), make no assumptions on the divisibility of n, and show that your bound is tight for infinitely many some values of n. (But part-marks may be given otherwise.) Justify your answer.

Hint: You might want to draw yourself a skip-list for $n = 4$ that satisfies the properties and verify that your bound holds for this n. Part-marks for this.

Question 3 [3 marks]

Let A be an unordered array with n distinct items $k_0, \ldots, k_{n-1}$. Give an asymptotically tight Θ-bound on the expected access-cost if you put A in the optimal static order for the following probability distribution:

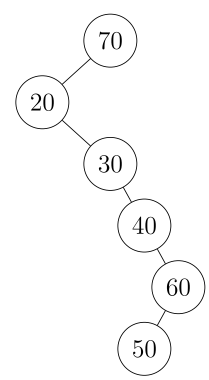
$$p_i = \frac{1}{(i+1)H_n} \text{ for } 0 \le i \le n-1 \text{ where } H_n = \sum_{j=1}^{n} \frac{1}{j}.$$

For example, for $n = 4$ we have $H_4 = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} = \frac{25}{12}$ and the items would have access probabilities $\langle \frac{12}{25}, \frac{6}{25}, \frac{4}{25}, \frac{3}{25} \rangle$.

Question 4 [1+2+9(+5)+4=16(+5) marks]

This assignment asks you to compare the performance of the MTF-heuristic for binary search trees with splay trees.

a) Consider the binary search tree shown on the right.



i) What is its potential function value when viewed as a splay tree? (State it with two fractional digits.)

ii) Show the binary search tree that results if you perform *splayTree::search*(50).

For both part-questions, it suffices to state the correct final answer but we recommend showing some intermediate steps so we can give part-marks in case of errors.

b) Let T be a binary search tree with n nodes and height $h = n - 1$, i.e., T is a path from the root to a unique leaf x. Show that if we perform *splayTree::search*(k) for the key k at x, then the resulting tree T' has height at most $h/2 + c$ for some constant c. Make c as small as possible.

Hint: Show a bound on the height of the subtree rooted at x after you have done i operations.

c) (Bonus) Create an example of a binary search tree T with n nodes and a sequence of $\Theta(n)$ operations *BST-MTF::search* for keys in T such that the total number of rotations is in $\Theta(n^2)$.

d) Prof. I.N.Correct claims that for any n they have an example of a binary search tree T with n nodes and a sequence of n operations *SplayTree::search* for keys in T such that the total number of rotations is in $\Theta(n^2)$. In particular the actual run-time for these n operations is in $\Omega(n^2)$.

Prove that this is impossible.

Question 5 [6 marks]

Recall *interpolation-search* (Algorithm 6.3 from the course notes) and consider its performance for the sorted array $A[0..n-1]$ where $A[i] = ai + b$ for $0 \leq i \leq n-1$ (for some constants $a > 0$ and b that are arbitrary real numbers). Show that then a search for a key k always takes $O(1)$ time, regardless of whether key k is in A or not.

Question 6 [8 marks]

This question concerns sorting a set of infinite-precision numbers $x_0, \ldots, x_{n-1}$. Specifically, each x_i is in $[0, 1)$ and written in base-2. It is given to you implicitly, via an accessor-function *get-decimal-place*(i, d), which returns the bit in the dth decimal place of x_i. For example, if $x_i = 0.001001...$ then *get-decimal-place*$(i, 3) = 1$ and *get-decimal-place*$(i, 4) = 0$. Function *get-decimal-place* takes $\Theta(1)$ time.

Describe an algorithm to sort these (implicitly given) numbers $x_0, \ldots, x_{n-1}$ in $O(n \log n)$ expected time, assuming the numbers $x_0, \ldots, x_{n-1}$ have been randomly and uniformly chosen from the interval $[0, 1)$. You may also assume that all numbers are distinct. Note that comparing x_i and x_j is *not* a constant-time operation! Your output should be the sorting-permutation π (i.e., $x_{\pi(0)} < x_{\pi(1)} < \cdots < x_{\pi(n-1)}$).

A high-level description is enough, no need for pseudo-code, and the correctness can be extremely short. (But do argue the run-time carefully.)