University of Waterloo
CS240E, Winter 2025
Assignment 3

Due Date: Tuesday, March 4, 2025 at 5pm

Be sure to read the assignment guideliness (`https://student.cs.uwaterloo.ca/~cs240e/w25/assignments.phtml#guidelines`). Submit your solutions electronically to Crowdmark.

Grace period: submissions made before 11:59PM on Mar. 4 will be accepted without penalty. Please note that submissions made after 11:59PM **will not be graded** and may only be reviewed for feedback.

Question 1 [5+7=12 marks]

A *Cartesian tree* is a treap except that the priorities can be arbitrary values (so are not necessarily a permutation of $\{0, \dots, n-1\}$). This means that we no longer have the array P that we had for treaps, i.e., we cannot look up the node that has a given priority.

Let T be a Cartesian tree that has height h and n nodes.

a) Describe an algorithm that is given T and a key k, and that prints all items in T for which the **key** is at least k. (There is no restriction on the order in which to print them.) Note that k need not exist as a key in T.

To state the desired run-time, we need a small detour. Let the *output-size* s be the number of items in T for which the key is at least k, and note that s could be anywhere between 0 and n. Your algorithm will have to spend $\Omega(s)$ time to print the items, and for this reason we include s in the desired run-time bound: Your algorithm should have $O(h + s)$ worst-case run-time.

b) Describe an algorithm that is given T and a priority p, and that prints all items in T for which the **priority** is at least p. (There is no restriction on the order in which to print them.) Note that p need not exist as priority in T. Your algorithm should have $O(1+s)$ worst-case run-time, where s is the output-size, i.e., the number of items where the priority is at least p.

Question 2 [5 marks]

Let S be a skip list that stores $n \geq 4$ non-sentinel keys. Assume that the lists $L_0, L_1, \dots, L_h$ of L have the following property for all $0 \leq i < h$:

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

(As in class, $|L_i|$ denotes the number of non-sentinels in list L_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 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 is tight for this n. Part-marks for this.

Question 3 [3 marks]

Let L be an unordered list with n distinct items $k_1, \ldots, k_n$. Give a Θ-bound on the expected access-cost if you put L in the optimal static order for the following frequencies:

$$f_i = \frac{n!}{i} \text{ for } 1 \le i \le n$$

For example, for $n = 4$ we have $n! = 24$ and the access-frequencies would be 24, 12, 8 and 6. Your bound must be in closed-form, simplify as much as possible.

Question 4 [2+2+2+5+4+2(+5)=17(+5) marks]

Consider the following algorithm to build a binary search tree:

Algorithm 1: *build-path-BST*(w, n)

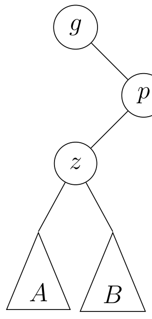
```
  Input: $w$ is a bit-string of length $n$
1 $T \leftarrow$ empty binary search tree; $\ell \leftarrow 0$, $r \leftarrow n-1$
2 for ($i = 0; i < n; i$++) do
3     if $w[i] = 0$ then  $k \leftarrow \ell; \ell$++ else $k \leftarrow r; r$−−
4     insert $k$ into $T$ using BST::insert (no rotations)
5 return ($T$)
```

a) Let $T(w)$ be the tree constructed for bitstring w of length n. Show that T has height $n - 1$, i.e., it is a path from the root to a unique leaf.

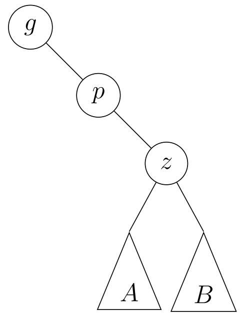
Let $T_{\mathcal{A}}(w)$ (for $\mathcal{A} \in \{\text{MTF}, \text{splay}\}$) be the binary search tree obtained as follows: First build $T(w)$, and let z be its unique leaf. Now apply rotations at z until it becomes the root. Here we use either the *MTF-heuristic* (i.e., single rotations) or the *splay-heuristic* (i.e., zig-zig and zig-zag rotations), and $\mathcal{A}$ indicates the applied heuristic.

b) (Warm-up) Assume that we have done rotations until z is the grandchild of the root, so up to symmetry the current tree is in one of the following formations:

(i)



(ii)



Show what the final tree looks like, for both formations and both heuristics. (So there are four trees to show in total.) No justification needed.

The following questions relate the height of $T_{\mathcal{A}}(w)$ to the height of $T(w)$ (which we know to be $h := n-1$ from part (a)). Let $H_{\mathcal{A}}(w)$ be the height of $T_{\mathcal{A}}(w)$ and as usual let $H^{\text{worst}}_{\mathcal{A}}(n)$, $H^{\text{best}}_{\mathcal{A}}(n)$ be the maximum/minimum of $H_{\mathcal{A}}(w)$ over all bitstrings of length n.

c) Show that $H^{\text{worst}}_{\text{MTF}}(n) = h$, by giving a suitable string w and showing $T(w)$, $T_{\text{MTF}}(w)$, and some intermediate steps of the transformation. Your construction should work for **any** n (give it for a small n of your choice and then explain how to generalize).

d) Show that $H^{\text{worst}}_{\text{splay}}(n) \leq h/2 + 1$ if n is odd (so h is even).

Hint: Bound the height of the subtree at z after i (zig-zig or zig-zag) rotations.

e) Show that $H^{\text{best}}_{\text{MTF}}(n) \geq \lceil h/2 \rceil$.

Hint: Consider the structure of the two subtrees of z after we have done some rotations.

f) Show that $H^{\text{best}}_{\text{splay}}(n) < h/2$, by giving a suitable string w and showing $T(w)$, $T_{\text{splay}}(w)$, and some intermediate steps of the transformation. If suffices to do this for **one** odd n (and try to keep it small, say $n \leq 17$).

g) (Bonus) Give an exact bound on $H^{\text{best}}_{\text{splay}}(n)$ that depends only on n and uses no asymptotics. In other words, state a function $f(n)$, and show that for all n that are divisible as needed, you have: (i) $T_{\text{splay}}(w) \geq f(n)$ for all bitstrings w of length n, and (ii) there exists a bitstring w of length n with $T_{\text{splay}}(w) \leq f(n)$.

Part-marks will be given if you show only one of the inequalities, as long as your $f(n)$ is close to the correct answer.

Question 5 [2+6=8 marks]

a) Consider the following array A of size 10:

0	1	2	3	4	5	6	7	8	9
10	20	30	40	50	60	70	80	90	100

Show the result of performing *interpolation-search*(A,10,55) in this array, by indicating for each round the current values of ℓ, r and the values used for the computation of m.

For your convenience, the pseudo-code of *interpolation-search* is given below.

Algorithm 2: *interpolation-search*(A, n, k)

```
$\ell \leftarrow 0$, $r \leftarrow n-1$
while ($\ell \leq r$) do
    if ($k < A[\ell]$) then return "not found, would be between indices $\ell - 1$ and $\ell$"
    if ($k > A[r]$) then return "not found, would be between indices $r$ and $r+1$"
    if ($k = A[r]$) then return "found at index $r$"
    $m \leftarrow \ell + \lceil \frac{k - A[\ell]}{A[r] - A[\ell]} \cdot (r - \ell - 1) \rceil$
    if ($A[m] = k$) then return "found at index $m$"
    else if ($A[m] < k$) then $\ell \leftarrow m + 1$
    else $r \leftarrow m - 1$
```

b) Consider a 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 *interpolation-search*(A, n, k) always takes $O(1)$ time, regardless of whether key k is in A or not.

Question 6 [5 marks]

This question involves a set $S = \{x_0, \ldots, x_{n-1}\}$ of infinite-precision numbers. Specifically, each x_i is in $[0, 1)$ and is 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. The numbers in S have been randomly and uniformly chosen from the interval $[0, 1)$ and are all distinct.

Give an algorithm that finds a longest common prefix among the numbers in S, i.e., a number $y = 0.y_1 y_2 y_3 \ldots y_s$ such that there are at least two numbers in S that begin with y, and for which s is as large as possible. (Note that y is not necessarily unique, but s is.) The worst-case run-time should be $O(sn)$, though significant partial credit will be given for an expected run-time of $O(n(s + \log n))$.