University of Waterloo
CS240E, Spring 2026
Written Assignment 3

Due Date: Tuesday, June 30, 2026 at 5:00pm

Be sure to read the assignment guideliness (`https://student.cs.uwaterloo.ca/~cs240e/s25/assignments.phtml#guidelines`). Submit your solutions electronically to Crowdmark. Ensure you have read, signed, and submitted the **Academic Integrity Declaration** AID02.

Grace period: submissions made before 7:59pm on June 30 will be accepted without penalty. Please note that submissions made after 7:59pm **will not be graded** and may only be reviewed for feedback.

Question 1 [8 marks]

Recall the idea of lazy deletion in a sorted array A: You mark items as 'deleted' (rather than actually removing them), and you rebuild the entire array if the number of 'deleted' items is bigger than the number n of non-deleted items in A. As a consequence of this rebuild, no items marked 'deleted' are left. The operations have the following actual run-times (you need not prove this).

- *search*: $\Theta(\log n)$
- *insert*: $\Theta(n)$
- *delete* without rebuilding: $\Theta(\log n)$.
- *rebuild* (as part of *delete*): $\Theta(n)$.

Show that the amortized run-time of *search* and *delete* is in $O(\log n)$, while the amortized run-time of *insert* is in $O(n)$.

Hint: you may use "the number of deleted items" as the potential function.

Question 2 [3+5=8 marks]

Recall that the Selection problem receives as input a set of n items and an integer k with $0 \leq k \leq n-1$ and it must return the item that would be at $A[k]$ if the items were put into an array A in sorted order.

(a) Argue that any comparison-based algorithm for Selection on n keys must have $\Omega(\log n)$ worst-case time.

(a) Let T be an scapegoat($\frac{2}{3}$)-tree that stores n items. Argue that `Selection`(T, k) can be done in $O(\log n)$ time.

Question 3 [2+6=8 marks]

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

a) Consider the following array A of size 10:

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 1: *interpolation-search*(A, n, k)

```
1 ℓ ← 0, r ← n − 1
2 while (ℓ ≤ r) do
3     if (k < A[ℓ]) then return "not found, would be between indices ℓ − 1 and ℓ"
4     if (k > A[r]) then return "not found, would be between indices r and r + 1"
5     if (k = A[r]) then return "found at index r"
6     m ← ℓ + ⌈ (k − A[ℓ]) / (A[r] − A[ℓ]) · (r − ℓ − 1) ⌉
7     if (A[m] = k) then return "found at index m"
8     else if (A[m] < k) then ℓ ← m + 1
9     else r ← 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 4 [4 marks]

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

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

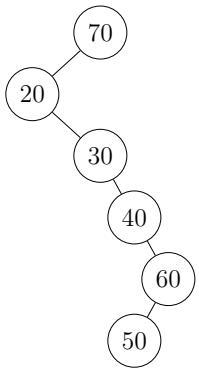
(As in class, $|L_i|$ denotes the number of non-sentinels in list L_i.) What is the maximum possible value of h, relative to n? For full marks, you should give an exact bound (no asymptotics), make no assumptions on the divisibility of n, and show that your bound is tight for infinitely many values of 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 5 [1+2+9+5+4=21 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) 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 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 $y = 0.y_1y_2y_3 \ldots y_s$ 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 $O(n(s + \log n))$.