York University
EECS 4101/5101, Winter 2023
Assignment 2

Due Date: February 12th, at 23:59

Paths are made by walking ...

Franz Kafka

All problems are written problems; submit your solutions electronically **only via Crowdmark**. You are welcome to discuss the general idea of the problems with other students. However, you must write your answers individually and mention your peers (with whom you discussed the problems) in your solution. There is also a bonus question, which is harder than other questions. It is recommended to approach this question only if you have completed other questions. Please refer to the course webpage for guidelines on academic integrity.

Problem 1 List-Update Algorithms [10 marks]

Consider the Move-By-Bit algorithm for the list update problem. Here, each item has a bit associated with it. In the beginning, all bits are 0. After access to an item x, Move-By-Bit moves x to the front if the bit of x is 1; otherwise, it keeps x at its position. In addition, after each access, the bit of the accessed item is flipped.
Use a potential function argument to show the competitive ratio of Move-By-Bit is at most 3. You should indicate your potential function and the amortized cost of the algorithm for different scenarios, and provide an upper bound for the ratio between the amortized cost of the algorithm and Opt . (In case you wonder, the actual competitive ratio of Move-By-Bit is 2.5.)

Hint: You must change the definition of potential. In doing so, you need to consider the bits of the two items involved in an inversion.

Problem 2 Compression [3+3+3+3 = 12 marks]

a) Apply the Burrows-Wheeler transform on the following string; show your work and the output.

HONEYPONY$

Assume $ precedes all characters when you sort rotations.

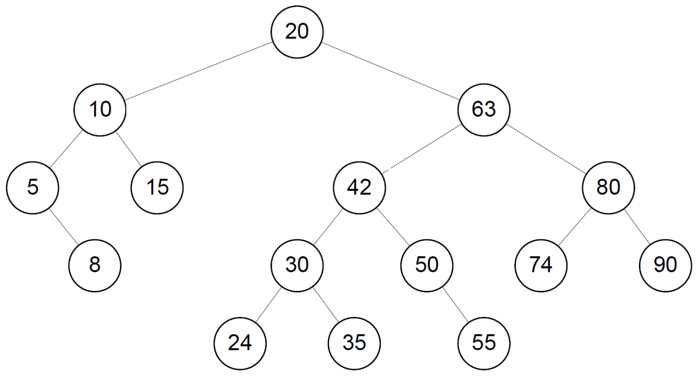
b) Assume an initial list $\$ \rightarrow A \rightarrow B \rightarrow E \rightarrow H \rightarrow N \rightarrow O \rightarrow P \rightarrow Y$, i.e., initially \$ is at index 0, A is at index 1, etc. Assume we use Move-To-Front on the above list to encode the outcome of the BWT transform from part (a). Show what numbers will be encoded (you need to show how the list is updated).

c) Assume an initial list $\$ \rightarrow A \rightarrow B \rightarrow C \rightarrow D$, i.e., initially \$ is at index 0, A is at index 1, etc. Assume we use Timestamp on the above list to encode $DB\$AABCD$. Show what numbers will be encoded (you need to show how the list is updated and, in particular, its last state).

d) Assume an initial list $\$ \rightarrow A \rightarrow B \rightarrow C \rightarrow D$. A compressing scheme that uses Move-To-Front has encoded the following numbers for a text T. Show what the actual text is. The numbers are 2 2 0 3 3.

Problem 3 AVL Trees [3+3+3+3 = 12 marks]

This problem will concern operations on the AVL tree T shown in Figure 1.



a) Show that T is an AVL tree by writing in the balance at each node.

b) Draw the tree after performing operation insert(33). Indicate any rotations that are required at each step.

c) Draw the *original tree* after performing operation delete(10). Swap with its *predecessor*. It suffices to draw the final tree.

d) Draw the *original tree* after performing operation delete(10). This time, swap with its *successor*. It suffices to draw the final tree.

Problem 4 (More) Binary Search Trees [8 marks]

We define *bar trees* as follows. A bar tree is a binary search tree where the heights of the left and right subtree for every node differ by at most 5. Prove that a bar tree with n nodes has height $O(\log n)$.

Problem 5 Augmentation [5 + 8 marks]

(a) Given a set of n KVPs with integer keys, describe a data structure that takes $O(n)$ space and supports the following two queries, both in $O(\log n)$ time:

- *range*(k_1, k_2) returns the number of KVPs in the dictionary whose key is in the range $(k_1, k_2]$. For example, for the dictionary with keys $\{1, 3, 4, 6, 7, 9, 13, 23\}$, the query *range*$(5, 12)$ must return 3.
- *rev-rank(k) returns the number of KVPs larger than k.*

(bonus) Consider a disk of circular shape, and suppose we are given n line-segments, with endpoints on the disk. Describe an time algorithm that runs in $O(n \log n)$ and reports the number of pairs of line-segments that intersect. Assume that no two line-segments share an endpoint. For example, in the left figure below, the outcome must be 3, and on the right figure, it must be 1.

Hint: Start from an endpoint, and form an array of size $2n$ of endpoints (labelled by their interval) by visiting endpoints in clockwise order. In the left example below, the resulting array will be $[1, 2, 1, 3, 4, 3, 2, 4]$, and for the right example, the array will be $[1, 2, 3, 3, 4, 2, 4, 1]$. Now, scan the array and maintain (an initially empty) augmented AVL tree while scanning the array.