Practice Sheet II

COL106: Data Structures and Algorithms

Semester-I 2023–2024

Problem I: Queue Using Two Stacks

Suppose you are given two stacks S_1 and S_2. Your task is to implement a Queue using only S_1 and S_2 and **no** other auxilliary memory.

1. Think of a design and explain how you will implement the `enqueue` and `dequeue` functions. Write psuedocode for these functions.

2. If you have thought of a good design, both your `enqueue` and `dequeue` function should have an $O(1)$ amortized cost. Prove this result, i.e. starting from an empty queue, the total time needed for N queue operations is $O(N)$.

Problem II: Modified MergeSort

Consider a variant of MergeSort where an input list of size n is divided into 3 sub arrays of size $n/2, n/4$ and $n/4$ respectively.

1. Write down the recurrence relation for the running time of the algorithm.

2. Solve the recurrence relation to obtain a bound on the running time of the algorithm. Is it any better than the original MergeSort?
   (*Hint:* One approach is to create a tree to unravel the recurrence. Try to check the amount of work done at each level of the tree.)

Problem III: Leaf Depths in Binary Trees

Suppose a binary tree has M leaves $l_1, l_2, ..., l_M$ at depths $d_1, d_2, \cdots, d_M$, respectively. Prove that

$$\sum_{i=1}^{M} 2^{-d_i} \leq 1$$

Problem IV: kth Largest Element in 2-4 Trees

You are given a 2-4 tree T, with the height $h = O(\log n)$, where n is the number of nodes in T. Let us say there are no duplicate keys in the 2-4 tree T. You will receive multiple queries, each query asking you to find the kth largest element of the 2-4 tree. Your task is two-fold:

- Augment the nodes of the 2-4 Tree with some auxillary data which will help with the queries
- Design an algorithm to find the kth largest element given the augmented tree.

Problem V: Merging AVL Trees

You are given two AVL Trees T_1 and T_2 with height h_1 and h_2 respectively. Further, you are given that all keys in the tree T_1 are smaller than all keys in T_2.

1. Design an algorithm which given T_1 and T_2, merges them and outputs a new AVL tree T_3.
2. Analyse the running time of your algorithm. An optimal approach would solve this problem in $O(h_1 + h_2)$ time.

Hint: Keep in mind that a rotation in an AVL Tree is $O(1)$, so you may consider using a constant number of rotations in your approach.