601.433/633 Introduction to Algorithms **Fall 2024**
Homework #4 **Due:** October 3, 2024, 9am

Remember: you may work in groups of up to three people, but must write up your solution entirely on your own. Collaboration is limited to discussing the problems – you may not look at, compare, reuse, etc. any text from anyone else in the class. Please include your list of collaborators on the first page of your submission. You may use the internet to look up formulas, definitions, etc., but may not simply look up the answers online.

Please include proofs with all of your answers, unless stated otherwise.

1 Amortized Analysis (34 points)

In this problem we have two stacks A and B (recall that a stack allows us to push elements onto it and to pop elements off of it in LIFO order). In what follows, we will use n to denote the number of elements in stack A and use m to denote the number of elements in stack B. Suppose that we use these stacks to implement the following operations:

- PushA(x): Push element x onto A.
- PushB(x): Push element x onto B.
- BigPopA(k): Pop $\min(n, k)$ elements from A.
- BigPopB(k): Pop $\min(m, k)$ elements from B.
- Move(k): Repeatedly pop one element from A and push it into B, until either k elements have been moved or A is empty.

We are using the stacks A and B as black boxes – you may assume that PushA, PushB, BigPopA(1), and BigPopB(1) each take one unit of time (i.e., it takes one time step to push or pop a single element).

Use a potential function argument to prove that the amortized running time of every operation is $O(1)$.

2 Amortized analysis of 2-3-4 trees (33 points)

Recall that in a 2-3-4 tree, whenever we insert a new key we immediately split (on our way down the tree) any node we see that is full (has 3 keys in it).

(a) (11 points) Show that in the worst case, the number of splits that we do in a single operation can be $\Omega(\log n)$. In other words, show that for there is a series of n inserts so that the next insert causes $\Omega(\log n)$ splits. You can be a bit informal here — just explain at a high level what such a sequence would look like and why it would result in $\Omega(\log n)$ splits.

Let's try to get around this worst-case bound by using amortized analysis. We will try to prove that the amortized number of splits is only $O(1)$. Note that this does not mean that the amortized running time of an insert is $O(1)$ (since this is not true); it just means that the amortized number

of splits is $O(1)$. So think of "cost" not as "time", but as "number of splits". Since we didn't talk about them in class, feel free to assume there are no delete operations.

(b) (11 points) Since we only have to split a node when it's full, a natural approach is to have a bank on every node which works as follows. When a node is first created its bank is initialized to 0. When a node becomes full, we add 1 to its bank. Then when we split a node we use the 1 in its bank to pay for the split. In other words, a node's bank is 1 if the node is full and 0 otherwise.

This argument unfortunately does not work. Prove that this banking scheme *does not* imply that the amortized number of splits is $O(1)$.

(c) (11 points) Show how to modify this argument to work. That is, show how we can use a bank at each node to prove that the amortized number of splits is $O(1)$. Hint: the previous part shows that this bank cannot just equal 1 if the node is full and 0 otherwise.

3 Union-Find (33 points)

In this problem we'll consider what happens if we change our Union-Find data structure to *not* use path compression. We will still use union-by-rank, but Find operations will no longer compress the tree. More formally, consider the following tree-based data structure. Every element has a parent pointer and a rank value.

Make-Set(x): Set $x \to parent := x$ and set $x \to rank := 0$.
Find(x): If $x \to parent == x$ then return x. Else return $Find(x \to parent)$.
Union(x, y):
Let $w := Find(x)$ and let $z := Find(y)$.
If $(w \to rank) \geq (z \to rank)$ then set $z \to parent := w$, else set $w \to parent := z$.
If $(w \to rank) == (z \to rank)$, set $(w \to rank) := (w \to rank) + 1$

In this problem we will analyze the running time of this variation.

(a) (11 points) Recall that the height of any node x is the maximum over all of the descendants of x of the length of the path from x to that descendant. Prove that for every node x, the rank of x is always equal to the height of x. Hint: use induction.

(b) (11 points) Prove that if x has rank r, then there are at least 2^r elements in the subtree rooted at x (we did this in class for the more complicated data structure which uses path compression, but now you should do it for this version without path compression).

(c) (11 points) Using the previous two parts, prove that every operation (Make-Set, Union, and Find) takes only $O(\log n)$ time (where n is the number of elements, i.e., the number of Make-Set operations).