# CS4311 Design and Analysis of Algorithms

Homework 4

Exam 2: May 11, 2022 (2 hours)

You are encouraged to work in groups. Please submit your solution (of any question) to our tutors on or before May 5, 2022, so that they may have a chance to give feedback of your work.

## I. Amortized Analysis

1. Bill has invented a strange data structure called Flipping Stack, which supports only `Flipping-Push` function. In each `Flipping-Push`, an item is first pushed to the stack; then, if the number of items in the stack is a power of 2, all items in the stack will have to be flipped. For instance, suppose we use `Flipping-Push` to push the items 1, 2, 3, 4 into the stack, the contents of the stack (viewed from bottom to top) after each `Flipping-Push` are as follows:

   $$(1) \Rightarrow (2,1) \Rightarrow (2,1,3) \Rightarrow (4,3,1,2)$$

   The cost of `Flipping-Push` is equal to 1 plus the number of items that are flipped. That is, if no flipping occurs, the cost is 1. Else, it is 1 + the number of items in the stack.

   Analyze the amortized cost of `Flipping-Push` using all the three methods taught in the class.

2. A min-heap with $n$ elements supports `Insert` and `Extract-Min` in $O(\log n)$ worst-case time. Give a potential function $\Phi$ such that the amortized cost of `Insert` is $O(\log n)$ and the amortized cost of `Extract-Min` is only $O(1)$. Show that your potential function works.

   Furthermore, explain why it is impossible to have the amortized cost of Insert to be $o(\log n)$ and at the same time the amortized cost of Extract-Min is $O(1)$.

   *Hint:* Sorting lower bound.

3. A sorted array allows efficient searching in logarithmic time. However, if we want to insert a new element, it requires linear time in the worst case to keep the array sorted. In fact, we can improve the insertion time (in the amortized sense) by making the searching time a bit slower.

   Let $n$ be the number of elements in the current array. Let $k = \lceil \log(n+1) \rceil$, so that the binary representation of $n$ has $k$ bits. Our scheme is to partition the $n$ elements into $k$ arrays $A_0, A_1, \ldots, A_{k-1}$ based on $n$'s binary representation. Precisely, let $\langle b_{k-1}, b_{k-2}, \ldots, b_0 \rangle$ be the binary representation of $n$, with $b_{k-1}$ the most significant bit. The array $A_i$ will hold $2^i$ elements in sorted order if $b_i = 1$; otherwise, it will be empty.

   (Does the scheme sound familiar? Looks like a binomial heap, right?)

   Based on this scheme, searching for an element will need to search all $k$ arrays in the worst case, and the total time will be $O(k \log n) = O(\log^2 n)$. To insert a new element, the number of elements, $n$, will be increased. Then, its binary representation will change, so we need to partition the elements in a different way.

   Describe how to insert a new element into this data structure so that the partitioning can be maintained correctly. Show that the amortized insertion time is $O(\log n)$.

## II. Binomial Heaps

1. Draw the structure of a binomial heap with 21 nodes.

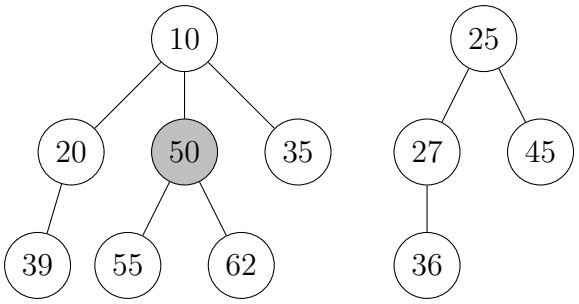
2. Suppose that we perform $n$ insertions on a binomial heap, starting from an empty one. Show that the total time is bounded by $O(n)$.

3. Suppose that we perform a sequence of $n$ make-heap, insert, or union operations on a collection of binomial heaps, starting from an empty collection. Show that the total time is bounded by $O(n)$.

## III. Fibonacci Heaps

1. Suppose that a root $x$ in a Fibonacci heap is marked. Explain how we can obtain such a scenario.

2. Peter has maintained a set of integers using Fibonacci Heaps, and the following is its current status. The marked nodes are colored, and the minimum pointer (not shown) is pointing at 10.




Peter is going to perform the following sequence of operations. Describe clearly how the Fibonacci Heaps will change after each operation.

(a) Decrease-Key 55 to 3

(b) Extract-Min

(c) Insert 18

(d) Decrease-Key 36 to 12

(e) Extract-Min