York University | EECS 4101 | June 17, 2023

Homework Assignment #5
Due: June 28, 2023 at 10:00 p.m.

[4] **1.** Prove or disprove the following statement. At any time, the node with the maximum degree in a Fibonacci heap is one of the roots.

2. A treap is a binary tree that stores a pair (k, p) in each node so that the k values satisfy the binary search tree property and the p values satisfy the min-heap-ordering property. We shall call the k values *keys* and the p values *priorities*.

[3] **(a)** Draw a treap containing the pairs $(4, 2), (7, 8), (2, 9), (5, 1), (1, 6), (9, 4)$.

[3] **(b)** Explain why there is a *unique* treap that contains the pairs $(k_1, p_1), \ldots, (k_n, p_n)$ if the priority values are all distinct.

[2] **(c)** Suppose we want to build a binary search tree containing keys $k_1, k_2, \ldots, k_n$. We choose priorities $p_1, p_2, \ldots, p_n$ uniformly at random from a very large range (so that the probability of any two priorities being the same is negligible). If you sorted the (k_i, p_i) pairs according to their priorities and then inserted the *keys* into a normal binary search tree in this order, explain why you would get a tree of exactly the same shape as the treap built from the pairs $(k_1, p_1), \ldots, (k_n, p_n)$.

[2] **(d)** Explain why the treap built using random priority values as described in part (c) has expected height $O(\log n)$.

Remark: you can build a treap by inserting elements one-by-one so that the time for each insertion is proportional to the height of the tree, as follows.

```
1: function TREAP-INSERT(T, k, p)
2:     create a new node z and set z.key ← k and z.priority ← p
3:     TREE-INSERT(T, z)
4:     while z.parent ≠ nil and z.priority > z.parent.priority do
5:         if z is the left child of its parent then RIGHT-ROTATE(T, z.parent)
6:         else LEFT-ROTATE(T, z.parent)
7:         end if
8:     end while
9: end function
```

The TREE-INSERT on line 3 is the standard BST insertion algorithm to insert the node z into T. The calls to the RIGHT-ROTATE and LEFT-ROTATE routines (which are described in Section 13.2 of the textbook) perform rotations that move z up to its parent's position.