1 Treap dimensions

Consider a treap with \( n \) distinct elements constructed by assigning random priorities to the elements.

(a) [5 points] Prove that the expected distance from the root of the treap to the smallest element is \( H_n - 1 \), where \( H_n = \sum_{i=1}^{n} 1/i \).

(b) [10 points] Prove that the expected distance from the root of the treap to a randomly picked element is \( 2 (\frac{n+1}{n}) H_n - 4 \).

(c) [Extra credit] Find the expected height of the root of the treap (the longest distance from root to any leaf).

(d) [5 points] In a perfectly balanced binary tree of \( n \) elements, show that the expected distance from the root to a random picked element is \( \log_2 n - \Theta(1) \).

2 Treaps and Set operations

(a) [6 points] Let \( A \) and \( B \) be two lists of sizes \( m \) and \( n \) (\( m \leq n \)), each containing elements drawn from the same completely ordered set. Elements within each list are distinct and completely sorted in ascending order (\( A \cap B \) is not necessarily \( \emptyset \)). Suppose that we wanted to merge \( A \) and \( B \) in to one completely ordered list \( C \). Show that at least \( \Omega(m \log(n/m)) \) comparisons are needed. Show the same lower bound for computing a sorted list which is the union of lists \( A \) and \( B \). Use the following definition for \( \Omega \) notation:
Given two functions \( f, g : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \), \( f(x, y) = \Omega(g(x, y)) \) if there exist \( m_0, n_0, c > 0 \) such that \( \forall m > m_0, n > n_0, f(m, n) \geq c \cdot g(m, n) \).

**b) [6 points]** Set operations like union, intersection and difference can be performed efficiently by storing the elements of the set in a random treap instead of a simple list. To do these set operations, two basic operations are useful:

- **Cut** \((x, T) \to (T_L, T_R)\): split a random treap \( T \), with priorities to each node assigned independently and randomly, in to two random treaps \( T_L \) and \( T_R \) so that all elements in \( T_L \) (\( T_R \)) are smaller (larger) than \( x \). If \( x \) is found in \( T \), the node containing \( x \) is also returned.

- **Splice** \((T_L, T_R) \to T\): given two random treaps \( T_L, T_R \) such that every element in \( T_L \) is smaller than every element in \( T_R \), merge them in to one treap.

Show how to do a **Cut** operation that results in two treaps of size \( n \) and size \( m \) in \( O(\log n + \log m) \) expected time. Also show how to do a **Splice** operation on treaps of size \( n \) and size \( m \) in \( O(\log n + \log m) \) expected time.

**c) [8 points]** Construct set union, intersection and difference operations for sets stored in treaps using the **Cut** and **Splice** operations.

**d) [Extra credit]** Show that the union of two random treaps of size \( m \) and \( n \), \( m \leq n \), can be computed in expected time \( O(m \log((n + m)/m)) \).

### 3 Splay Trees

**a) [5 points]** Recall the rules of splay\((x, S)\):

(i) if \( x \) has a parent but no grandparent, just rotate\((x)\).

(ii) if \( x \) has parent \( y \) and a grandparent, and if \( x \) and \( y \) are either both left children or both right children, first rotate\((y)\) then rotate\((x)\).

(iii) if \( x \) has parent \( y \) and a grandparent, and if one of \( x, y \) is a left child and the other is a right child, first rotate\((x)\) and then rotate\((x)\) again.

Now we propose a new set of rules by replacing rules (ii) and (iii) with “if \( x \) has a parent \( y \) and a grandparent, first rotate\((x)\) and then rotate\((x)\) again.” How does the running time using this new set of rules compare to the original one? Provide your analysis.

**b) [8 points]** For (b), (c), please read Prof. Sleator’s notes:

http://www.cs.cmu.edu/afs/cs/academic/class/15750-s01/www/notes/lect0123-splay
Suppose we can assign arbitrary positive values to the weights of the nodes. For each assignment of weights we can derive a bound on the running time of a sequence of accesses. By assigning the frequently accessed items a high weight, we will be able to get tighter bounds on the running time.

Let \( w(x) > 0 \) be the weight of a node \( x \). Let \( s(x) = \sum_{y \in T_x} w(y) \) and \( T_x \) be the subtree rooted at \( x \). The Access Lemma states that, using a potential of \( \log(s(x)) \) for each node, the amortized cost of each access \( \sigma_i \) is

\[
1 + O \left( \log \frac{s(\text{root})}{s(\sigma_i)} \right).
\]

Come up with a weight assignment \( w : \{1, \cdots, n\} \mapsto \mathbb{R}_{>0} \) that proves that for a fixed target \( f \in \{1, \cdots, n\} \), the cost of a sequence of accesses \( \sigma = \sigma_1\sigma_2\cdots\sigma_m \) in a splay tree storing the keys \( \{1, \cdots, n\} \) is

\[
O(m + n \log n + \sum_{i=1}^{m} \log(|f - \sigma_i| + 1))
\]

(Hint: use the Access Lemma)

(c) [7 points] Prove (or give a counter example) that the running time of a sequence of accesses \( \sigma = \sigma_1\sigma_2\cdots\sigma_m \) in a splay tree is \( O(m + n \log n + \sum_{i=1}^{m} \min_{1 \leq j \leq n} \log(j + |f_j - \sigma_i|)) \) where \( (f_1, \cdots, f_n) \) is an arbitrary permutation of the keys \( \{1, \cdots, n\} \).