1 Lazy Deletion

Suppose a student has implemented a balanced binary search tree (e.g., AVL-tree, red-black tree, etc.) that performs insertion and search operations in $O(\log n)$ time, but was too lazy to implement deletion. Instead, they have implemented a lazy deletion mechanism: To delete an item, we search for the node that contains it (in $O(\log n)$ time) and then mark that node as deleted. When the number of marked nodes exceeds the number of unmarked nodes (during a deletion) the entire tree is rebuilt (in $O(n)$ time) so that it contains only unmarked (i.e., undeleted) nodes.

1. Define a non-negative potential function and use it to show that the amortized cost of deletion is $O(\log n)$.

2. How does your potential function affect the amortized cost of insertion?
2 Lazy Insertion Data Structures

Suppose we have a static data structure for some search problem. Given \( n \) elements, we can build a data structure in \( O(n \log n) \) time that answers queries in \( O(\log n) \) time. We convert this into an insertion-only data structure as follows: We maintain two static data structures, \( D_1 \) has size at most \( \sqrt{n} \) and \( D_2 \) has size at most \( n \). To insert a new element we first check if the number of elements in \( D_1 \) is less than \( \sqrt{n} \). If so, we add the newly inserted element to \( D_1 \) and rebuild \( D_1 \) at a cost of \( O(\sqrt{n} \log n) \). Otherwise (there are too many elements in \( D_1 \)) we take all the elements of \( D_1 \), move them to \( D_2 \) and rebuild \( D_2 \) at a cost of \( O(n \log n) \). To search for an element, we search for it in both \( D_1 \) and \( D_2 \) at a cost of \( O(\log n + \log \sqrt{n}) = O(\log n) \).

1. Define a potential function on \( D_1 \) and \( D_2 \) to show that the amortized cost of insertion is \( O(\sqrt{n} \log n) \).

2. Show that during the second case of the insertion procedure, even if we only insert half the elements of \( D_1 \) into \( D_2 \), the amortized cost of insertion is still only \( O(\sqrt{n} \log n) \).
3. Suppose we generalize this data structure so that we maintain $d$ static data structures $D_1, \ldots, D_d$ where $D_i$ has maximum size $n^{i/d}$. Whenever $D_i$ becomes full we empty it and put all its elements in $D_{i+1}$.

Define a potential function on $D_1, \ldots, D_d$ to show that the amortized cost of insertion is $O(n^{1/d} \log n)$. 

3 Array-Based Priority Queues

In this question, we investigate an implementation of priority queues based on a collection of sorted lists. In this implementation we store $O(\log n)$ sorted lists $L_0, \ldots, L_k$, where the list $L_i$ has size at most $2^i$. To find the minimum element, we simply look at the first element of each list (remember, they are sorted) and report the minimum, so the operation $\text{FindMin}$ takes $O(\log n)$ time.

1. To do an insertion, we find the smallest value of $i$ such that $L_i$ is empty, merge the new element as well as $L_0, \ldots, L_{i-1}$ into a single list and make that list be $L_i$. At the same time, we make $L_0, \ldots, L_{i-1}$ be empty.

Prove, by induction on $i$, that the list $L_i$ has size at most $2^i$.

2. Show how to merge $L_0, \ldots, L_{i-1}$ so that the cost of this merging (and hence the insertion) is $O(2^i)$. 

3. Starting with an empty priority queue and then performing a sequence of \( n \) insertions, how many times does list \( L_i \) go from being empty to being non-empty.

4. Using your answer from the previous question, what is the total running time of a sequence of \( n \) insertions beginning with an empty priority queue?

5. As this data structure evolves, the elements move to lists with larger and larger indices. Define a non-negative potential on the element \( x \) so that when \( x \) is in \( L_0 \) its potential is \( \log n \) and when \( x \) is in \( L_{\log n} \) its potential is 0.
6. Define a non-negative potential function on this data structure so that, when we build the list \( L_i \), the potential decreases by at least \( c2^i \), for some constant \( c \).

7. Show that the amortized cost of insertion (using your potential function from the previous question) is \( O(\log n) \).