Design and Analysis of Algorithms

Practice-sheet : **Amortized Analysis**

1. **Deleting half elements**

   Design a data structure to support the following two operations for a dynamic multiset S of integers, which allows the duplicate values:

   - *Insert*(S, x): inserts x into S.
   - *Delete-Larger-Half*(S): delete the largest $\lceil |S|/2 \rceil$ elements from S.

   Explain how to implement this data structure so that any sequence of m *Insert* and *Delete-Larger-Half* operations run in $O(m)$ time. Your implementation should also include a way to output the elements of S in $O(|S|)$ time.

   Hint: You might like to use the following result: there exists an $\mathcal{O}(n)$ time algorithm to compute median of a set of n numbers.

2. **Alternate potential functions**

   We discussed the algorithm for the fully dynamic table in the class. Using a specific potential function, we showed that the amortized cost of each operation is $O(1)$. For each of the following potential functions, verify whether it will also ensure $O(1)$ amortized cost for each insert/delete operation ?

   (a) $c(4n - size(T))$

   (b) If $n \geq size(T)/2$ then $c(2n - size(T))$; else $c(size(T)/2 - n)$.

   Hint: Just verify whether these functions satisfy the properties of a valid potential function and lead to the desired amortized cost. For (b), see what happens at and around $n = size(T)/2$.

3. **Extract-min in $O(1)$ time**

   Consider an ordinary binary min-heap data structure with n elements supporting the operations *Insert* and *Extract-min* in $O(\log n)$ worst case time. Give a potential function ϕ such that the amortized cost of *Insert* is $O(\log n)$ and the amortized cost of *Extract-min* is $O(1)$.

   Hint: Just focus on the tree structure of the binary heap. What are the changes in this structure during these operations? What is the maximum depth of a node in the tree?

4. **Simulating a queue using stacks**

   Show how to implement a queue with two ordinary stacks so that amortized cost of each *Enqueue* and each *Dequeue* operation is $O(1)$.

   Hint: You will use one stack where elements will be inserted, and the other stack from where the elements will be removed. Under what circumstances will you have to transfer all elements of a stack to the other stack?

5. **Binary Search under insertions**
   Binary search of a sorted array takes logarithmic search time, but the time to insert a new element is linear in the size of the array. We can improve the time for insertion by keeping several sorted arrays. The following paragraph gives an overview of this data structure.

   The aim of the data structure is to support **search** and **insert** on a set elements. The set is initially empty. Let n be the number of elements in the set at any stage of time. Let $k = \lceil \log(n+1) \rceil$, and let the binary representation of n be $\langle n_{k-1}, n_{k-2}, ..., n_0 \rangle$. We have k sorted arrays $A_0, A_1, ..., A_{k-1}$, where for $i = 0, 1, ..., k-1$, the length of A_i is $2^i \cdot n_i$. Each array is either full or empty, depending upon whether $n_i = 1$ or $n_i = 0$, respectively. The total number of elements held in all k arrays is therefore $\sum_{i=0}^{k-1} 2^i = n$. Although each individual array is sorted, elements in different arrays bear no relationship to each others.

   Provide the details for answering search queries and performing insertion on the data structure described above. The time spent in performing n insert operations has to be $O(n \log n)$. The time taken for any search should be $O(\log^2 n)$.