Design and Analysis of Algorithms (CS345A)

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.

2. **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 $O(1)$.

3. **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) $c(4n - \textbf{size}(T))$

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

4. **Credit based analysis for dynamic table**

   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 $O(1)$. Use a credit based analysis for the algorithm and show that the actual time complexity of any sequence of n operations (insert/delete) will be $O(n)$.

   3c 4c

5. **Dynamic table with worst case $O(1)$ time per insertion**
Though we are able to achieve amortized $O(1)$ time per insertion for dynamic tables and maintain space utilization a constant (0.5 in this case), it is not always practical. Remember it takes $O(n)$ time when we insert into a full table (copying all the n elements into another table of double the size). There may be real world applications where we need *quick* response time. So is it possible to achieve worst case $O(1)$ time per insertion while still maintaining space utilization factor a constant > 0. where need Design an algorithm for dynamic table under insertion of elements such that the following constraints are satisfied.

   - At any stage of time, there has to be a table that should contain all the elements.
   - Total space utilization should be greater than a positive constant factor (independent of the number of elements).
   - The worst case time of inserting any element should be $O(1)$.

   Hint: Current approach (copying all elements into another table when it becomes full) is a *lazy* approach. Shed laziness and plan properly.

6. **Amortized analysis of Binary heap**
Recall binary heap that you might have studied during ESO207A. Let n denote the number of elements in a binary heap H By selecting a suitable potential function, carry out the amortized analysis of the operations on heap so that

   (a) Amortized cost of Extract_Min(H) is 0.
   (b) Amortized cost of Insert(H, x) is $O(\log n)$.

   Hint: Just consider the binary tree structures after each operation. Can you spot something *decreasing* after Extract_Min(H) ?

7. **Using arrays for binary search of a dynamic set**
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. Design a data-structure which is a collection of arrays only that can support any sequence of n insertions in $O(n \log n)$ time. The worst case time for the search is $O(\log^2 n)$.
Hint: Get inspiration from a binary counter, especially the way the bits are flipped during an increment operation.