Data Structure & Algorithms
CS210A,ESO207A,ESO211
Semester I, 2012-13, CSE, IIT Kanpur

Theoretical Assignment I

Deadline : 24th August

Note: The exercises in this assignment are extensions of the problems which we discussed in full details in the lectures till now. Therefore, you are advised to fully internalize the concepts and fundamentals of the corresponding lectures before attempting these exercises. Make sincere attempts to solve these exercises. In case you are stuck, you may contact the instructor before 20th August to get some hint or pointer (no penalty for such hints). But make sure you think really hard enough for each of these problems. Your solution for each exercise must be formal and complete. In particular,

- The design of each algorithm or data structure must be provided in full details.
- You must provide analysis of time and space complexity of each algorithm and data structure.
- You must prove the correctness of each algorithm as well.

1 Maximum Sum Submatrix

In the class, we discussed an $O(n)$ time algorithm for maximum-subarray problem : Given an array A storing n numbers, compute the subarray of A which has the maximum sum among all possible subarrays. Now your job is to design an efficient algorithm for an extension of this problem to 2-dimensions. Suppose you are given a $n \times n$ matrix M storing numbers. You have to design an $O(n^3)$ algorithm to find a sub-matrix whose sum is maximum sum over all possible submatrices of M. See the example shown in Figure 1 for a better understanding.

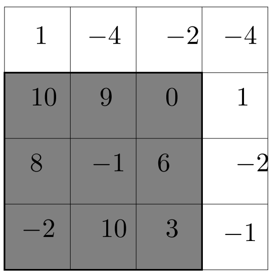

Figure 1: The shaded submatrix is the maximum sum sub-matrix

2 Finding k-majority Element

Recall the 2-majority problem discussed in the class. We developed an $O(n)$ time algorithm for this problem. Your goal now is to generalize it to the case of k-majority problem defined as follow.

Given an array A storing n elements and a parameter k which is a positive integer, an element of A is said to be k-majority of A if it appears more than $\frac{n}{k}$ times in A. Design an algorithm which computes a k-majority element, if exists, in array A. The algorithm must have $O(nk)$ time complexity. You can use $O(k)$ extra space. You can not modify the original array A even temporarily.

3 Range-Minima Problem

Recall the range-minima problem discussed in full details in Lecture 6. We developed a data structure for this problem which occupies $O(n \log n)$ space and answers each query in $O(1)$ time. Suppose, our application where we want to deploy our Range-minima data structure is very conservative about space. Our aim is to decrease the space at the expense of increased query time. So you are asked to design a data structure which should occupy $O(n)$ space and should be able to answer any range-minima query in $O(\log n)$ time. The design of your data structure has to proceed along the following lines:

For an array of size n, the data structure discussed in the class takes $O(n \log n)$ space. How and to what extent should we shrink the array suitably so that this data structure takes $O(n)$ space. Note that you may spend $O(\log n)$ time per query instead of $O(1)$.