Data Structure & Algorithms
CS210A
Semester I, 2014-15, CSE, IIT Kanpur

Practice Sheet 3: Divide and Conquer Paradigm

August 30, 2014

1. We define an element to be k-majority element if it appears more than n/k times in an array. In the class, we dicussed an $O(n)$ time algorithm for finding 2-majority element.
   (a) Use the concepts of this algorithm to design an $O(n)$ time algorithm to compute any 3-majority element ?
   (b) Design an $O(n \log n)$ time algorithm for finding 2-majority element which is based on divide and conquer paradigm in way similar to Merge Sort or Counting Inversion.
2. There is an array A storing n distinct numbers. Design an algorithm to find the maximum and the minimum of an array in $< 2n$ number of comparisions. Your algorithm may change array A if required.
3. In class, we discussed an $O(n)$ time algorithm for finding a contiguous subarray in a 1-d array which has the maximum sum. Though we were able to solve the problem efficiently, there exists a divide and conquer approach for the same problem which takes $O(n \log n)$ time. Design the divide and conquer algorithm, stating clearly the divide and conquer steps.
4. There are 2 sorted arrays A and B, each storing n distinct numbers. Write an algorithm to find the median of $A \cup B$. The time complexity of your algorithm should be $O(\log n)$.
5. Recall QuickSort algorithm discussed in the class. This algorithm is based on procedure **Partition**(A, ℓ, r) that rearranges this subarray so that all elements smaller than $A[\ell]$ precede $A[\ell]$ and all elements greater than $A[\ell]$ succeed $A[\ell]$. Design an $O(r - \ell)$ time implementation of **Partition**(A, ℓ, r) that uses only $O(1)$ extra space.
6. The running time of Quick Sort depends upon the element we choose for partition in each recursive call. Use this observation to find out the worst case running time of Quick Sort on an array of n elements? What can be the best case running time of Quick Sort ?
7. There are n points on real line. A is an array that stores the distance of each of these points from origin. Design an $O(n \log n)$ time algorithm to find the closest pair of points. In the next course CS345, we shall solve this problem when points are in a plane.