Home Work 4: Algorithm Design

Deadline: Friday, December 6, 2013: 4pm

P-1 [Marks 15] Searching

Given an array A of n integers and an integer X, find indices i and j, where $1 \leq i, j \leq n$, and $i \neq j$, such that, $X = A[i] + A[j]$. The $O(n^2)$ algorithm is obvious. Give an algorithm that does this in $O(nlgn)$. The algorithm returns (0, 0) if no (i, j) satisfy the summation condition.

P-2 [Marks 15] Decision problem

Give an $O(n \lg n)$ time algorithm that determines whether two sets S_1 and S_2 of n numbers each, are disjoint or not. (Two sets are disjoint if their intersection is an empty set). This problem is very similar to the previous problem.

P-3 [Marks 50] Maximum Sub-array Problem (CLRS 3rd Edition, Topic 4.1)

Given an array A of n real numbers (both positive and negative possible), find the maximum sum of any contiguous subarray. Write an algorithm to accomplish this task in $O(nlgn)$. Note that the $O(n^2)$ algorithm is obvious. For example, if A is $\{1, 5, -2, 4, 5, -4, 2, -10\}$, the required subarray is $\{1, 5, -2, 4, 5\}$.

P-4 [Marks 15] Searching

You have an array A of n integers which was first sorted and then rotated to the right k times. So if A was $\{1, 2, 3, 4, 5, 6, 7, 8\}$ and k was 3, then A became $\{6,7,8,1,2,3,4,5\}$. Given this rotated array (you do not know the value of k); write an $O(lgn)$ time algorithm to find the maximum element in A. You could try finding an $O(n)$ algorithm first.

P-5 [Marks 80] Closest Pair Problem: Computational Geometry (CLRS 3rd Edition, Topic 33.4)

Give an O(nlgn) algorithm to find out a closest pair of points in a set of n (2-D) points. The input is two arrays X and Y each of size n. The output should be the indices of two points that are closest. X contains the x-coordinates of the points where Y contains the corresponding y-coordinates of the points i.e. the k^{th} point is (X[k], Y[k]). Use Euclidian distance formula to compute distance between any two points.

P-6 [Marks 30] Quick-Sort Partitioning

Design an array of 20 distinct elements that makes the quick-sort run in the worst way if the pivot is selected to be the 4^{th} element of the array always. Dry run by fixing a particular in-place partitioning method.

P-7 [Marks 50] Quick-Sort with Repeated Elements

Design an O(n) in-place partitioning algorithm that handles non-distinct elements. If the pivot has multiple occurrences, then all elements that are smaller than the pivot should occur to the left and all elements that are larger than the pivot should occur to the right of the pivot. All the occurrences of the pivot should appear consecutive after partitioning. Now the output of the procedure is 2 indices, which are the indices of the left most and the right most occurrences of the pivots respectively. Modify the code of quick-sort to handle arrays that has repeated elements.

P-8 [Marks 30] Reductions

Like the decision tree argument for proving sorting to be $\Omega(n \lg n)$, it has been proven already that "Element uniqueness problem" has also the same lower bound as sorting i.e. $\Omega(n \lg n)$. Using the fact that "Element uniqueness problem" is $\Omega(n \lg n)$, prove that "Finding Mode Problem" is also $\Omega(n \lg n)$.