ESO 207

Practice Problems - 2

Devendra Agrawal Rachit Nimavat

September 2, 2014

1. The maximum subarray problem is to find the contiguous subarray within a one-dimensional array of numbers (containing at least one positive number) which has the largest sum. For example, for the sequence of values -2, 1, -3, 4, -1, 2, 1, -5, 4; the contiguous subarray with the largest sum is 4, -1, 2, 1, with sum 6. Given an array of integers, find Maximum sum subarray using divide and conquer technique in $\mathcal{O}(n)$ time. (***Hint*:** It may be helpful to design a divide-and-conquer technique where division is straightforward and the conquer phase requires $O(n)$ time. This will result in $O(n \log n)$ time algorithm. Next, the above algorithm can be improved by keeping track of some more quantities for each of the halves, resulting in the conquer phase to run in time $O(1)$. This gives the recurrence equation of $T(n) = 2T(n/2) + O(1)$ whose solution is $T(n) = O(n)$.)

2. Suppose you are consulting for a bank that is concerned about fraud detection,and they come to you with the following problem. They have a collection of n bank cards that they have confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic strip with some encrypted data, and it corresponds to a unique account number in the bank. Each account can have many bank cards corresponding to it, and we'll say that two cards are equivalent if they correspond to the same account. It is difficult to read the account number off a bank card directly, but the bank has a high tech "equivalence tester" that takes two bank cards and, after performing some computations, determines whether they are equivalent.
   The question is the following: among the collection of n cards, is there a set of more than $n/2$ of them that are equivalent to one another? Assume that the only feasible operations you can do with the cards are to pick two of them and plug them

into equivalence tester. Design a divide and conquer based algorithm to decide the answer of their question with only $\mathcal{O}(n\ logn)$ invocations of the equivalence tester. (***Note*:** The problem also has a solution with $O(n)$ invocations of the equivalence tester.).

3. Let us call a pair (i, j) a significant inversion if $i < j$ and $A[i] > 2A[j]$. Design an $\mathcal{O}(n\ logn)$ time algorithm to count the number of significant inversions in an array.

4. You are given a binary heap H of size n, a number x, and a positive integer k. Design an $\mathcal{O}(k)$ time algorithm to determine if x is smaller than k^{th} smallest element in H. You may use $\mathcal{O}(k)$ extra space.

5. Given a list of integers, find the n-th smallest number, i.e., the number that appears at index n (0-based) when they are sorted in non-descending order. The numbers will be given in intervals. For example, the intervals $(1, 3)$ and $(5, 7)$ represent the list of numbers $1, 2, 3, 5, 6, 7$. A number may be present in more than one interval, and it appears in the list once for each interval it is in. For example, the intervals $(1, 4)$ and $(3, 5)$ represent the list of numbers $1, 2, 3, 3, 4, 4, 5$.

6. Find the problem on spoj archive, link is `http://www.spoj.com/problems/PRO/`.

7. Find the problem on spoj archive, link is `http://www.spoj.com/problems/EXPEDI/`.