NCM IST, Mathematics for Computer Science
Problems in sorting and searching

18 June, 2018

1. Given two sets A and B of n numbers each, give an efficient algorithm to report all the elements that are common to both A and B. If the integers in A are in the range 1 to 100, can you design a better algorithm?

2. Given two sets A and B of n integers each, and a query integer x, give an efficient algorithm to determine whether $x = a + b$ for some $a \in A$ and $b \in B$. If the integers in A are in the range 1 to 100, can you design a better algorithm?

3. Let A be an array of N integers out of which the first n of them are positive and the remaining are negative. You don't know n.

   - Give an $O(\log N)$ algorithm to compute n.
   - Give an $O(\log n)$ algorithm to compute n.

4. Given an integer a and a positive integer n, describe a method to compute a^n using only multiplications. How many multiplications does your algorithm use? How many does it use to compute a^{100}?

5. Given a list of n numbers, give an efficient algorithm to determine whether they are all distinct.

6. Given a list of $2n$ distinct numbers, give an efficient algorithm to determine whether the list can be grouped into n groups of two elements each such that the sum of the pairs in each group is the same.

7. Give an algorithm to find the second largest element from a list of n numbers. Focus on minimizing the number of comparisons (Hint: there is an algorithm that takes $n + o(n)$ comparisons).

8. Suppose I modify a given sorted list of $4n$ distinct numbers as follows:

   Keep elements in even positions (positions 2, 4, 6, ... $4n$) as they are. Form n disjoint pairs (i, j) on the odd numbered positions where $i = 2k + 1$ for some $k = 0$ to $n - 1$ and $j = 2k + 1$ for some $k = n$ to $2n - 1$.

   Now swap elements in positions i and j for every such pair. (I.e. every element in an odd numbered position in the first half of the array is swapped with *some* element in an odd numbered position in the second half of the array. No element is involved in more than one swap (i.e. the swaps are disjoint). You don't know these (i, j) pairs except that an element in an odd numbered position in the first half of the array is swapped with *some* element in an odd numbered position in the second half.

   Now given an element x, explain how you can determine whether or not x is in the (new reshuffled) array in $O(\log n)$ time.