Data Structures and Algorithms (CS F211) – T2

Prof.N.L.Bhanu Murthy

BITS Pilani
Hyderabad Campus

Problem 1

Suppose that you are given a sorted sequence of *distinct* integers. Give an algorithm to determine whether there exists an i index such as $a_i = i$.

For example, in $\{-10, -3,3,5,7\}$, $a3 = 3$. In $\{2,3,4,5,6,7\}$, there is no such i.

Problem 1

Suppose that you are given a sorted sequence of *distinct* integers. Give an algorithm to determine whether there exists an *i* index such as $a_i = i$.

For example, in { − 10, − 3,3,5,7}, $a3 = 3$. In {2,3,4,5,6,7}, there is no such *i*.

```
Assume set indexes are zero based
FindIndex(A):
    1. low =  0, high =1
    2. mid = (low + high)/2
    3. if(A[mid] > mid) then
            Index will lie in left half of the array
                high = mid-1
                Go to step 2.
        else if (A[mid] < mid) then
            Index will lie in right half of array
            low = mid + 1
        else
            return mid
```

Problem 2

Suppose that you are given a sorted sequence of *distinct* integers

drawn from 1 to *m* where $n < m$. Give an algorithm to find an integer that is not present in *a*.

Problem 3

Suppose there are n distinct integers where in each is having values between 0 and n. Find out the missed out integer between 0 and n without sorting the input array.

Problem 4

Given two sorted arrays with sizes m and n then devise an algorithm to merge two lists into a single sorted list.

Problem 5

Let *M* be an n X m integer matrix in which the entries of each row are sorted in increasing order (from left to right) and the entries in each column are in increasing order (from top to bottom). Give an efficient algorithm to find the position of an integer *x* in *M*, or to determine that *x* is not there. How many comparisons of *x* with matrix entries does your algorithm use in worst case? $O(n + m)$ is necessary and sufficient. Lower bound comes from potentially independent values along second diagonal -- upper bound comes from observing that we can eliminate either a row or a column in each comparison if we start from the lower left corner and walk up or left.

Problem 5

Let *M* be an n X m integer matrix in which the entries of each row are sorted in increasing order (from left to right) and the entries in each column are in increasing order (from top to bottom). Give an efficient algorithm to find the position of an integer *x* in *M*, or to determine that *x* is not there. How many comparisons of *x* with matrix entries does your algorithm use in worst case? $O(n + m)$ is necessary and sufficient. Lower bound comes from potentially independent values along second diagonal -- upper bound comes from observing that we can eliminate either a row or a column in each comparison if we start from the lower left corner and walk up or left.

```
int row = 0, col = m-1;
while (row < n && col >= 0) {
    if (M[row][col] == key) {
        return new Point(row,col);
    }
    else if (M[row][col] < key) {
        col--;
    }
    else row++;
}
return NULL;
```

Problem 6

You have 10 bags full of coins, in each bag are 1,000 coins. But one bag is full of forgeries, and you can't remember which one. But you do know that a genuine coins weigh 1 gram, but forgeries weigh 1.1 grams. To hide the fact that you can't remember which bag contains forgeries, you plan to go just once to the central weighing machine to get ONE ACCURATE weight. How can you identify the bag with the forgeries with just one weighing?

Problem 6

You have 10 bags full of coins, in each bag are 1,000 coins. But one bag is full of forgeries, and you can't remember which one. But you do know that a genuine coins weigh 1 gram, but forgeries weigh 1.1 grams. To hide the fact that you can't remember which bag contains forgeries, you plan to go just once to the central weighing machine to get ONE ACCURATE weight. How can you identify the bag with the forgeries with just one weighing?

Sol:

If there is only 1 bag with forgeries, then take 1 coin from the first bag, 2 coins from the second bag . . . 10 coins from the tenth bag and simply weigh the picked coins together !

If there were no forgeries, you know that the total weight should be (1+2+3+ . . . +10) = 55 grams.

But if, for example, the weight is 55.3 grams, then you know that 3 coins are forgeries, so that must be bag 3. So, that solves it.

Problem 7

Take a sequence of 2*n* real numbers as input. Propose a procedure that partitions the numbers into *n* pairs, with the property that the partition minimizes the maximum sum of a pair. For example, say we are given the numbers (1,3,5,9). The possible partitions are ((1,3),(5,9)), ((1,5),(3,9)), and ((1,9),(3,5)). The pair sums for these partitions are (4,14), (6,12), and (10,8). Thus the third partition has 10 as its maximum sum, which is the minimum over the three partitions.

Can you think ways of improving the procedure from the perspective of number of operations used in the procedure..

Problem 8

Given an array of numbers. Give a procedure for checking whether there are any duplicated elements in the array or not? How many comparisons are required in the procedure? If the array is sorted array, can you improvise the procedure with lesser number of comparisons.

Problem 9

Given n real numbers, write down a procedure to find out a pair of elements that have maximum absolute difference between the two values. Also write down a procedure to pick up a pair of elements that have minimum absolute difference between the two values. Can you think of a better procedure to achieve the same with respect to the number of operations.

Thank You!!