Bonus Problems 1

Varun Goyal

Problem 1

Prove that any comparison-based convex hull algorithm needs to make $\Omega(n \log n)$ steps. (Recall that an algorithm is comparison-based if its **if**-statements use comparisons, for example comparing angles of two lines to find which angle is smaller, or comparing an angle with 180 degrees to see if the angle is obtuse.)

Solution

1. The first step to solve convex hull algorithm is to sort the points using the angles formed by the lines drawn between 3 points (presuming that one point is inside the said polygon)
2. As seen in the lectures and studied any "comparison based" sorting algorithm would take $\Omega(n \log n)$ time to be completed.
3. Rest all other steps of convex hull algorithm takes $\Omega(n)$ time.
4. Thus least time for any comparison based convex hull algorithm will be $\Omega(n \log n)$.
    a. However if we do-not use comparison based algorithm and instead use radix sort to sort the angles, the convex hull algorithm could be solved in $\Omega(n)$ time. (but then it would not be comparison based)

Problem 2

Read the description of Problem 2 from Homework 2. Imagine the DVD stacks are arranged in a circle, not a line. In other words, on the right of stack n is stack 1. Design an algorithm for this problem that is faster than $O(n2)$. Sketch the proof of correctness of your algorithm and estimate its running time.

Solution

ROBOT(arr[], n)

1. maxDvD = SELECT (arr[], n)
2. left = SUM (arr[0], arr[i_{maxDvD}])
3. right = SUM (arr[i_{maxDvD}], arr[n])
4. if (left > right)
    1. from i_{maxDvD} to n to i_{maxDvD} // i.e. from i_{maxDvD} to i_{maxDvD}-1 clockwise circle
        1. take access DvDs move to right.
5. else
    1. from i_{maxDvD} to 0 to i_{maxDvD} // i.e. from i_{maxDvD} to i_{maxDvD}-1 anti-clockwise circle
        1. take access DvDs move to left.

Proof Of Correctness:

I would call it a SeeSaw Water Flow Algorithm say if water is accumulated at the center of a seesaw in a lot of quantity and then it is let to flow it will flow in the direction where there is a slope towards the ground.

In this algo, the place where max dvds are stacked will be the center of the seesaw and the slope is towards where the sum of dvds is lowest.

Running Time:

O(n)

Problem 3

We are given an array *A* with *n* elements and a number *C*. Assume that the sum of the elements in *A* is larger than *C*. We would like to compute the smallest number of elements from *A* whose sum is at least *C*. For example, if *A* = [8*;* 3*;* 9*;* 2*;* 7*;* 1*;* 5] and *C* = 18 then the answer is 3. Design an $O(n)$ algorithm for this problem and argue its correctness and running time.

Solution

3. Value =0 // is the final number of elements which is the solution to the problem

Smallest Element(A,n,C,Value)

a) P = Select(A,n/2) //using select algorithm covered in the class lectures

b) Sum= A[n/2 +1]+A[n/2 +2]+…….+ A[n]

c) if Sum > C

B = A[n/2+1], A[n/2 +2],……..A[n] //creating a sub array

Smallest Element(B,n/2, C,Value)

Else

If Sum < C

Sum = Sum –C

B = A[1],A[2],……A[n/2]

Value = Value + n/2

Smallest Element(B,n/2,Sum,Value)

Else if Sum = C ,return Value.

The algorithm makes use of the fact that after finding the median all the values created in the array after the median are greater than it and before the median are smaller than it.

The values grater than median are summed and compared with C, if found less than those numbers are to be included and c = sum –c is passed again to the algorithm

If sum of elements (greater than median) is greater than c then only the right half is to be considered and that is passed to the function as we have to check the largest (first) k elements such that their sum just exceeds C

The algorithm runs in O(n) which can be proved using recurrence as

$T(n) = O(n) + T(n/2) + \text{constant}$