Assignment 4

Algorithms & Complexity (CIS 522-01)

Javier Arechalde

February 28, 2018

Part A: Read the solved exercises and Practice

Solved excercise #1 in Chapter 5

In this problem, we have an array with n entries, and inside this array, we have a peak entry p in a position j of the array, so that the values in the array prior to j go in increasing order, and after the peak value, they go in decreasing order.

Our goal is to find that peak entry p without having to read the entire array, and only by reading as few values as possible.

Algorithm Pseudocode

Algorithm 1 Finding maximum pseudocode

function FINDMAXIMUM($pos_{start}, pos_{end}, array$)
    $n = (pos_{start} + pos_{end})/(2)$
    if $array(\frac{n}{2} - 1) < array(\frac{n}{2}) < array(\frac{n}{2} + 1)$ **then**
        We have a **positive slope**, so we havent reached the maximum yet
        FINDMAXIMUM($(pos_{start} + pos_{end})/2, pos_{end}, array$)
    else if $array(\frac{n}{2} - 1) > array(\frac{n}{2}) > array(\frac{n}{2} + 1)$ **then**
        We have a **negative slope**, we already passed the maximum
        FINDMAXIMUM($pos_{start}, (pos_{start} + pos_{end})/2$)
    else if $array(\frac{n}{2} - 1) > array(n/2) < array(\frac{n}{2} + 1)$ **then**
        We have found the **maximum point**
        return value(n/2)
    end if
end function

Solution for problem instance of size 10

In this case, we will run our algorithm, when we have an instance of size 10.

In this problem's scope, we will have an set of increasing numbers that grow until a maximum, followed by another set of numbers that go in decreasing order.

This will be our problem's working set:

$$S = [1, 2, 4, 12, 14, 21, 6, 4, 3, 1]$$

If we start running the algorithm, we will check the number in the 5^{th} position.

In this case we will have that $S[4] < S[5] < S[6]$, which means that we are in a positive slope, and we still haven't reached the maximum. Then we will call the function again, so in the next iteration of our algorithm, we will work with this set.

$$S' = [14, 21, 6, 4, 3, 1]$$

Now we will check the 3^{rd} position. In this case we have that $S[2] > S[3] > S[4]$, which means that we are in a negative slope, so we already passed the maximum. Thus, we will call the function again, and in the next iteration of our algorithm we will work with this set.

$$S'' = [14, 21, 6]$$

Now we will check the middle position, in this case the 2^{nd} position. We get the result that $S[1] < S[2] > S[3]$, which means that the number we are checking is indeed the maximum. Now we will return that value, and stop running our algorithm.

$$max = 21$$

Time Complexity

In this problem, with each one of the recursive calls, we reduce the problem to one of at most half the size of the initial problem. Then:

$$T(n) \leq T(n/2) + c$$

when $n > 2$, and

$$T(2) \leq c.$$

Then, the running time of our algorithm will be $O(\log n)$.

Solved exercise #2 in Chapter 5

In this problem, we have an investment company that looks at n consecutive days of a given stock. For each of these days, the stock has a price $p(i)$ per share for the stock on that day. We assume that the stock prize was fixed on that day.

The goal is to find, without having to check each possible combination of days, which will take $O(n^2)$, in which day they should have bought the shares, and in which day they must have sold them to make as much money as possible.

Algorithm Pseudocode

Algorithm 2 Stocks Divide-and-Conquer pseudocode

```
function Max(list)
    max
    for element in list do
        if element > max  or  max == NA then
            max = element
        else
            continue
        end if
    end for
    return max
end function

function Min(list)
    min
    for element in list do
        if element < min  or  min == NA then
            min = element
        else
            continue
        end if
    end for
    return min
end function

function FindOpt(array)
    if len(array) > 2 then
        Larray = array[0 : len(array)/2]
        Rarray = array[len(array)/2 : len(array)]

        LSide = FindOpt(Larray)
        RSide = FindOpt(Rarray)

        LOpt = LSide[1] - LSide[0]
        ROpt = RSide[1] - RSide[0]
        MOpt = Max(RSide) - Min(LSide)

        Marray = [min(LSide),max(RSide)]

        maxvalue = Max(LOpt,ROpt,MOpt)
        if maxvalue == LOpt then
            return LSide
        else if maxvalue == ROpt then
            return RSide
        else if maxvalue == MOpt then
            return Marray
        end if

    else if len(array) <= 2 then
        return array
    end if
end function
```

Solution for problem instance of size 10

Now we will compute a solution using our algorithm for a problem instance of size 10.

This will be the set we are going to work with.

$$S = [7, 4, 3, 2, 5, 7, 10, 20, 14, 7]$$

First, as the array is longer than 2, we will divide it into two sets recursively until we have arrays of length smaller or equal to 2.

$$[7, 4, 3, 2, 5] \quad [7, 10, 20, 14, 7]$$

$$[7, 4] \quad [3, 2, 5] \quad [7, 10] \quad [20, 14, 7]$$

$$[7, 4] \quad [3, 2] \quad [5] \quad [7, 10] \quad [20, 14] \quad [7]$$

Now we are going to compare set by set, if the optimal solution between one set, compare it with the other set, and compare it also with the maximum of the "right" set and the minimum of the "left" set, and see which one gives us a better solution.

$$[7, 4] \quad \& \quad [3, 2]$$

$$OptL = 4 - 7 = -3 \quad OptR = 2 - 3 = -1 \quad OptM = 3 - 4 = -1$$

$$[5] \quad \& \quad [7, 10]$$

$$OptM = 10 - 5 = 5 \quad OptR = 3$$

$$[20, 14] \quad \& \quad [7]$$

$$OptL = -6 \quad OptM = 7 - 14 = -7$$

We return $[3, 2] \quad [5, 10] \quad [20, 14]$.

Now we have:

$$[3, 2] \quad [5, 10] \quad [20, 14]$$

Lets start comparing.

$$[3, 2] \quad \& \quad [5, 10]$$

$$OptL = -1 \quad OptR = 5 \quad OptM = 10 - 2 = 8$$

We return $[2, 10]$.

Now we have:

$$[2, 10] \quad \& \quad [20, 14]$$

$$OptL = 8 \quad OptR = -6 \quad OptM = 18$$

We finally return $[2, 20]$, and that will be our final result.

Time Complexity

As we said, in this problem, we will have to recursively subdivide our array, and take the best possible solution out of this three possible solutions:

- The optimal solution on S
- The optimal solution on S'
- the optimal solution of $p(j) - p(i)$, over $i \in S$ and $j \in S'$.

The first two items are computed, in time $T(n/2)$ by recursion, and the third item is computed by finding the maximum in S' and the minimum in S, which can be done in $O(n)$ time. Then our running time $T(n)$ satisfies

$$T(n) \leq 2T(\frac{n}{2}) + O(n)$$

,

Then the time complexity of our implementation will be $O(n \log n)$.

Part B: Problem Solving

Significant inversion

Problem Model

In this problem, we are given a sequence of n numbers $a_1, ..., a_n$, which we will we assume that are all distinct, and we define inversion to be a pair $i < j$ such that $a_i > a_j$. We call a pair *significant inversion* if $i < j$ and $a_i < 2a_j$.

Our goal is to count the number of *significant inversions* between two orderings, using an algorithm that has $O(n \log n)$ time complexity.

Pseudocode

Algorithm 3 Significant inversion pseudocode

Running time

Local minimum

Problem Model

In this problem, we are give a complete binary tree T. Each node v of T is labeled with a real number x_v. For each node in the tree, we can only determine its value x_v by probing the node v.

Our goal is to find a local minimum, that is if the label x_v is less than the label x_w for all the nodes w that are joined to v by an edge. We also have to find this local minimum of Tusing only $O(logn)$ probes to the nodes of T.

Pseudocode

Algorithm 4 Local minimum pseudocode

Implementation

Running time