Courtney Gosselin
Greg Petsul
Andrew Swan
Tasha Kucher
COSC 320
Dr. Yong Gao
Nov 30, 2018

Project Report COSC 320

Problem Formulation:

In this project, we intend to compute the day trader's strength for a given time period. This is similar to the maximum subarray problem, which aims at finding the maximum contiguous subarray. A day trader makes his money daily by buying and selling stocks within the same trading day and sells them all at the end of it make a profit or loss. We are to construct an algorithm that calculates the day traders performance over a time period. Now we are going to look at this problem as if we were given a sequence of n integers and need to determine the maximum subsequence. That is the maximum sum over all possible contiguous subsequences. Below is the algorithmic problem we constructed before we begun.

Algorithmic Problem:

FindADayTradersStrength()

- I - profits[ ] - complete trading record (the profit on each trading day) each index representing the day(s) profit.
- Q- Find the strength S of a day trader which is the maximum performance over all possible time periods.

Design and Analysis:

Descriptions of algorithms A and B:

Algorithm A is straightforward, first the performance of the trader of all possible time periods, and then take the maximum of these values as the strength S.

Algorithm B first creates an array that contains all possible performance times (time length being the index) for every given time period and sets to zero if below. Then the algorithm just finds the max of the created array. If however the max is equal to zero

(which is in the case it does equal zero or that the array only had negative values) the max is set to the base case (zero).

Pseudocode of algorithms A and B:

```
public int algorithmAforStrength(int[] profits){
     max ← profits[0]
     localSum ← 0
     N ← profits.length

     //i = size of interval
     //j = starting index in interval
     //k = current position in current interval


     for(i ← 0 to n){
          for(j ← 0 to n-1){
               for(k ← j; k to (j+1-1)){
                    localSum += profits[k]
               }
               if(localSum is larger than max){
                    max ← localSum
               }
               localSum ← 0
          }
     }
     return max
}

public int algorithmBforStrength(int[] profits){
          int n ← profits.length        //set n to size of array
          int [] x ← new int[n]         //create array
          x[0] ← profits[0]
          int max ← 0
          for (int i ← 1 to n){
               if(x[i-1] < 0)
                    x[i-1] ← 0
               x[i] ← profits[i] + x[i-1]

               if(x[i] > max)
                    max ← x[i]
          }

          return max
     }
```

Formal Analysis:

Asymptotic Notation:

The runtime of A is: $O(n^3)$ This runtime is easily seen because of the three for loops used to come to the conclusion.

The runtime for B is: O(n) Since the algorithm is only traversing the for loop once.

Runtime:

The runtime of A is: $O(n^3)$

The runtime for B is: O(n).

Algorithm B (O(n)) is significantly more efficient than Algorithm A ($O(n^3)$)

Proof of Correctness (Induction)

Prove by **induction**.

Maximum = max{x[i-1]+profits, Maximum}

Base case: n <= 1. Trivially true, either the algorithm will output 0 for n=0 or profits[0] for n=1

Assume true for n-1

Strength[n] will always be correct because we know the n^{th} day will either be part of our max or it won't. If it is part of our max then then new max would be x[i-1]+profits[i] (x[i-1] being our current total profit value in our current stretch of concurrent days(local max) and profits[i] being our profits for that given day) otherwise our final max at n days remains at max (our max strength over all days up to n (strength[n-1]).

Testing:

The below results tracks the time in each of algorithm return real time run time. For 8000, 500k, and multi-million records. For the runtime of algorithm A which is cubic time and for algorithm B it is runs at linear time.

Results:

When there are 8000 records algorithm A runs at ~25 seconds and B runs at 0 seconds.

When there are 500,000 records algorithm A runs at *Too long* and B runs at 0 seconds.

When there are multi-million records algorithm A runs at *Too long* and B runs at 0 seconds.

Conclusion

This project allowed us to work as a team and implement both the brute-force approach as well as converse and work as a team as we came to the conclusion of a dynamic approach. Taking the problem and relating it to the subarray problems we did in class as well as the dynamic programming we did in assignment 2. We were able to create a linear time algorithm.