Timothy Allec

Name, SID, Date .....

In Class Assignment 5: Maximum Subarray

Benjamin Sanders, MS November 25, 2020

1 Introduction

You will need to work individually to complete this assignment. Write your name at the top of all pages for this assignment. Turn in all work to Blackboard on or before the deadline to receive credit.

You may use additional libraries and online resources, if you get them approved in writing, over email, from the instructor first. If you have received approval from the instructor, write the approved libraries and any references in the space below.

import java.util.Vector;

.....
.....
.....
.....

2 Assignment Description

2.1 Big Picture

This algorithm is used by stock market analysts to understand when optimal buy and sell points exist in the history of a given stock, regardless of whether it is a Bull or Bear market.

2.2 Algorithm Implementation

Implement the following algorithm in Java, using the Vector data structure for any 1-D array, 2-D array, or linear algebra purposes.

The initial call FIND-MAXIMUM-SUBARRAY$(A, 1, A.length)$ will find a maximum subarray of $A[1..n]$.

FIND-MAXIMUM-SUBARRAY$(A, low, high)$

```
1  if high == low
2      return (low, high, A[low])          // base case: only one element
3  else mid = ⌊(low + high)/2⌋
4      (left-low, left-high, left-sum) =
           FIND-MAXIMUM-SUBARRAY(A, low, mid)
5      (right-low, right-high, right-sum) =
           FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
6      (cross-low, cross-high, cross-sum) =
           FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
7      if left-sum ≥ right-sum and left-sum ≥ cross-sum
8          return (left-low, left-high, left-sum)
9      elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10         return (right-low, right-high, right-sum)
11     else return (cross-low, cross-high, cross-sum)
```

Name, SID, Date .....

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4      sum = sum + A[i]
 5      if sum > left-sum
 6          left-sum = sum
 7          max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11      sum = sum + A[j]
12      if sum > right-sum
13          right-sum = sum
14          max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

2.3 Time Complexity Analysis

Where n is the number of data points in A, analyze the time complexity of the given algorithm with respect to n. Write the result of your analysis in big-O notation, i.e. $O(n^2)$ in the space below.

O(n^2)

.....

2.4 Space Complexity Analysis

Where n is the number of data points in A, analyze the space complexity of the given algorithm with respect to n. Write the result of your analysis in big-O notation, i.e. $O(n \cdot log(n))$ in the space below.

O(n)

.....

Name, SID, Date .....

2.5 New Algorithm Design and Implementation

In the space below, design an algorithm that complements the given algorithm. Research how MAXIMUM-SUBARRAY can be used for a stock trader. The purpose is to find optimal buy and sell points in a stock's history.

Use pseudocode written in a style similar to the given algorithm, and implement it in Java. You may use as many additional pages as necessary for this purpose.

```
import java.util.Vector;

public class StockTrader {
   // FindMaximumSubarray method modified to return buy and sell points for stock trading
   public Vector<Integer> FindMaximumSubarray(Vector<Integer> A, int low, int high) {
      if (high == low) {
         Vector<Integer> baseCase = new Vector<Integer>();
         baseCase.add(low); // Buy point
         baseCase.add(high); // Sell point
         return baseCase; // base case: only one element
      } else {
         int mid = (low + high) / 2;
         Vector<Integer> leftResult = FindMaximumSubarray(A, low, mid);
         Vector<Integer> rightResult = FindMaximumSubarray(A, mid + 1, high);
         Vector<Integer> crossResult = FindMaxCrossingSubarray(A, low, mid, high);

         // Compare results and return the one with the maximum sum
         if (A.get(leftResult.get(0)) <= A.get(rightResult.get(0))
               && A.get(leftResult.get(1)) >= A.get(rightResult.get(1))
               && A.get(leftResult.get(1)) >= A.get(crossResult.get(1))) {
            return leftResult;
         } else if (A.get(rightResult.get(1)) >= A.get(leftResult.get(1))
               && A.get(rightResult.get(0)) <= A.get(leftResult.get(0))
               && A.get(rightResult.get(1)) >= A.get(crossResult.get(1))) {
            return rightResult;
         } else {
            return crossResult;
         }
      }
   }
```

```
// FindMaxCrossingSubarray method modified to return buy and sell
points
   public Vector<Integer> FindMaxCrossingSubarray(Vector<Integer>
A, int low, int mid, int high) {
      int leftSum = Integer.MIN_VALUE;
      int sum = 0;
      int maxLeft = mid;

      // Find the maximum sum on the left side
      for (int i = mid; i >= low; i--) {
         sum = sum + A.get(i);
         if (sum > leftSum) {
            leftSum = sum;
            maxLeft = i;
         }
      }

      int rightSum = Integer.MIN_VALUE;
      sum = 0;
      int maxRight = mid + 1;

      // Find the maximum sum on the right side
      for (int j = mid + 1; j <= high; j++) {
         sum = sum + A.get(j);
         if (sum > rightSum) {
            rightSum = sum;
            maxRight = j;
         }
      }

      // Return buy and sell points with maximum sum
      Vector<Integer> result = new Vector<Integer>();
      result.add(maxLeft); // Buy point
      result.add(maxRight); // Sell point
      return result;
   }
         public static void main(String[] args) {
            // Example usage:
            StockTrader stockTrader = new StockTrader();

            // Create a sample vector representing stock prices
            Vector<Integer> stockPrices = new Vector<>();
            stockPrices.add(7);
            stockPrices.add(1);
            stockPrices.add(5);
            stockPrices.add(3);
            stockPrices.add(6);
            stockPrices.add(4);

            // Call FindMaximumSubarray method
            Vector<Integer> result =
      stockTrader.FindMaximumSubarray(stockPrices, 0,
      stockPrices.size() - 1);

            // Display the result
            System.out.println("Optimal Buy and Sell Points: [" +
      result.get(0) + ", " + result.get(1) + "]");
         }
      }
```

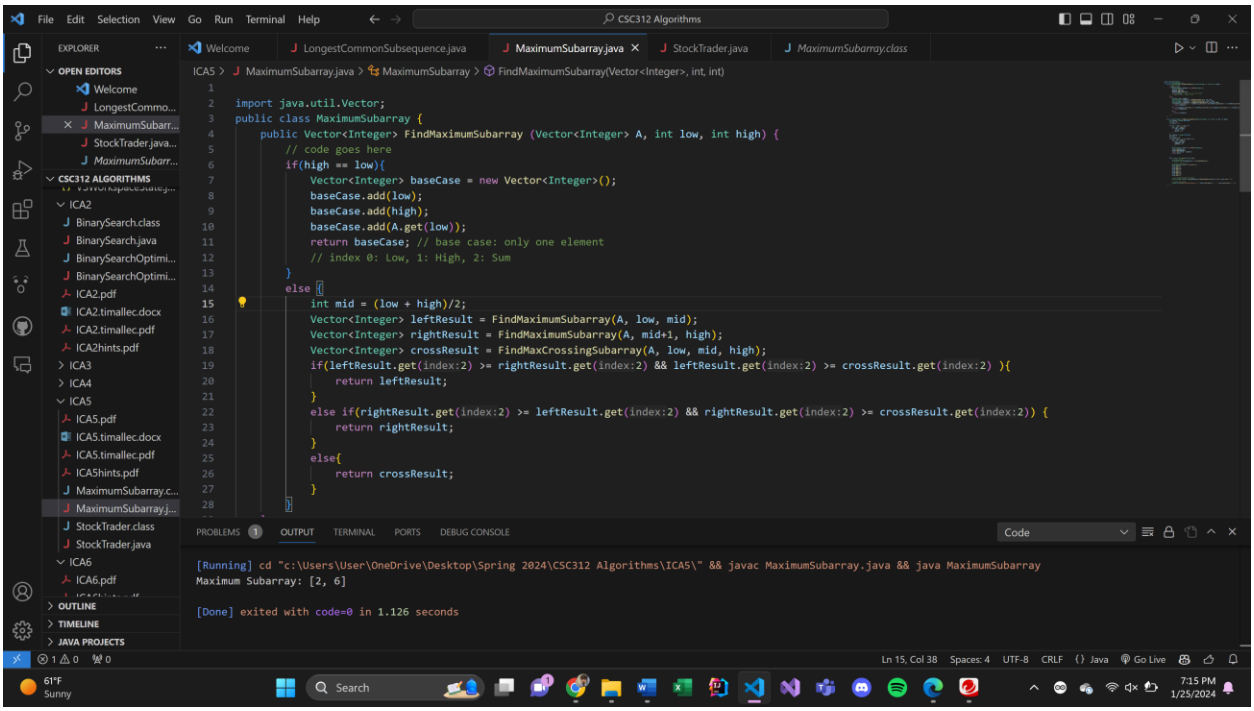
3 What to Turn In

Turn in one PDF or Word document on Blackboard, containing the following items.

1. All pages scanned or photographed of the In Class Assignment completed document.
2. Any additional pages you used to complete the assignment.
3. All code created for the assignment, along with test cases.
4. One statement indicating which parts of your implementation(s) are working, and which parts are not.
5. Screenshots demonstrating the code working, if it is working.

```java
import java.util.Vector;
public class MaximumSubarray {
    public Vector<Integer> FindMaximumSubarray (Vector<Integer> A, int low, int
high) {
        // code goes here
        if(high == low){
            Vector<Integer> baseCase = new Vector<Integer>();
            baseCase.add(low);
            baseCase.add(high);
            baseCase.add(A.get(low));
            return baseCase; // base case: only one element
            // index 0: Low, 1: High, 2: Sum
        }
        else {
            int mid = (low + high)/2;
            Vector<Integer> leftResult = FindMaximumSubarray(A, low, mid);
            Vector<Integer> rightResult = FindMaximumSubarray(A, mid+1, high);
            Vector<Integer> crossResult = FindMaxCrossingSubarray(A, low, mid,
high);
            if(leftResult.get(2) >= rightResult.get(2) && leftResult.get(2) >=
crossResult.get(2) ){
                return leftResult;
            }
            else if(rightResult.get(2) >= leftResult.get(2) && rightResult.get(2)
>= crossResult.get(2)) {
                return rightResult;
            }
            else{
                return crossResult;
            }
        }
    }

    public Vector<Integer> FindMaxCrossingSubarray(Vector<Integer> A, int low,
int mid, int high){
        int leftSum = Integer.MIN_VALUE;
        int sum = 0;
        int maxLeft = 0;

        for(int i = mid; i >= low;i--){
            sum = sum + A.get(i);
            if ( sum > leftSum){
                leftSum = sum;
```

```
                maxLeft = i;
            }
        }
        int rightSum = Integer.MIN_VALUE;
        sum = 0;
        int maxRight = 0;
        for(int j = mid +  1; j <= high; j++){
            sum = sum + A.get(j);
            if (sum > rightSum){
                rightSum = sum;
                maxRight = j;
                }
            }
            Vector<Integer> result = new Vector<Integer>();
            result.add(maxLeft);
            result.add(maxRight);
            result.add(leftSum + rightSum);
            return result;


        }
        public static void main(String[] args) {
            // Example usage:
            MaximumSubarray maximumSubarray = new MaximumSubarray();

            // Create a sample vector
            Vector<Integer> array = new Vector<>();
            array.add(-2);
            array.add(-3);
            array.add(4);
            array.add(-1);
            array.add(-2);
            array.add(1);
            array.add(5);
            array.add(-3);

            // Call FindMaximumSubarray method
            Vector<Integer> result = maximumSubarray.FindMaximumSubarray(array,
0, array.size() - 1);

            // Display the result
            System.out.println("Maximum Subarray: [" + result.get(0) + ", " +
result.get(1) + "]");
        }
    }
```

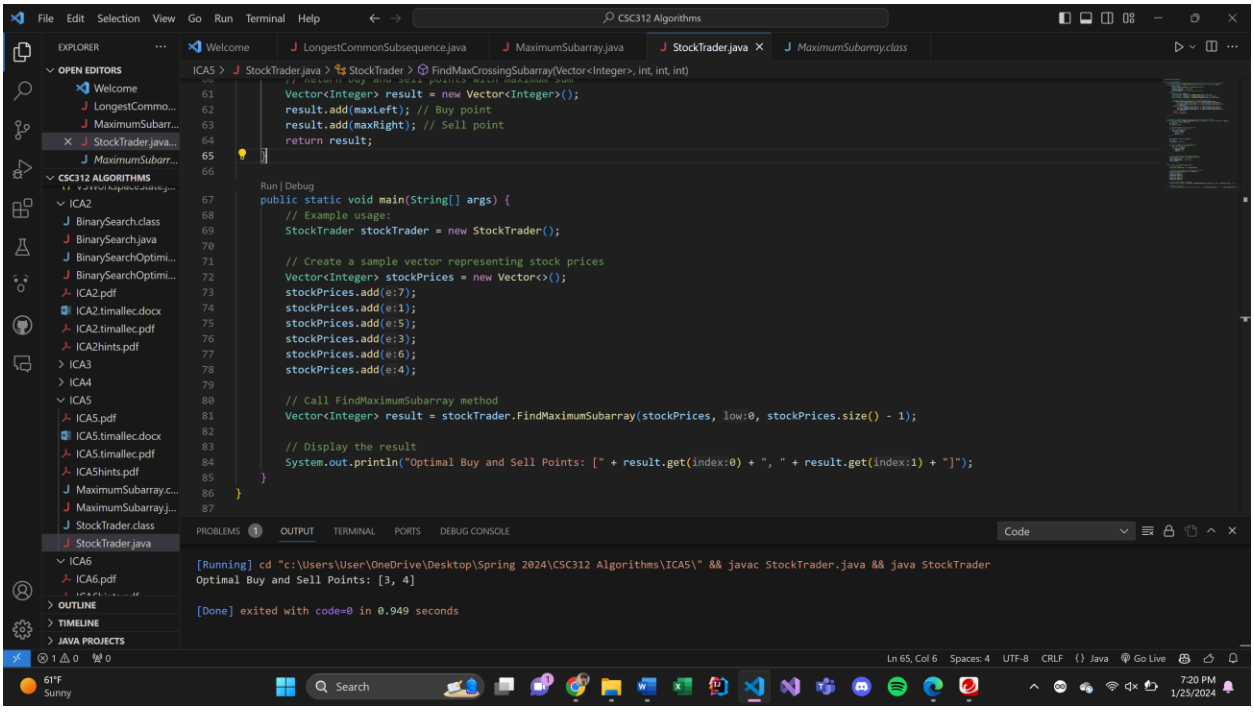


OPTIMIZATION “STOCK TRADER”

```
import java.util.Vector;

public class StockTrader {
    // FindMaximumSubarray method modified to return buy and sell points for
stock trading
    public Vector<Integer> FindMaximumSubarray(Vector<Integer> A, int low, int
high) {
        if (high == low) {
            Vector<Integer> baseCase = new Vector<Integer>();
            baseCase.add(low); // Buy point
            baseCase.add(high); // Sell point
            return baseCase; // base case: only one element
        } else {
            int mid = (low + high) / 2;
            Vector<Integer> leftResult = FindMaximumSubarray(A, low, mid);
            Vector<Integer> rightResult = FindMaximumSubarray(A, mid + 1, high);
            Vector<Integer> crossResult = FindMaxCrossingSubarray(A, low, mid,
high);

            // Compare results and return the one with the maximum sum
            if (A.get(leftResult.get(0)) <= A.get(rightResult.get(0))
                    && A.get(leftResult.get(1)) >= A.get(rightResult.get(1))
                    && A.get(leftResult.get(1)) >= A.get(crossResult.get(1))) {
                return leftResult;
            } else if (A.get(rightResult.get(1)) >= A.get(leftResult.get(1))
```

```
                    && A.get(rightResult.get(0)) <= A.get(leftResult.get(0))
                    && A.get(rightResult.get(1)) >= A.get(crossResult.get(1))) {
                return rightResult;
            } else {
                return crossResult;
            }
        }
    }

    // FindMaxCrossingSubarray method modified to return buy and sell points
    public Vector<Integer> FindMaxCrossingSubarray(Vector<Integer> A, int low,
int mid, int high) {
        int leftSum = Integer.MIN_VALUE;
        int sum = 0;
        int maxLeft = mid;

        // Find the maximum sum on the left side
        for (int i = mid; i >= low; i--) {
            sum = sum + A.get(i);
            if (sum > leftSum) {
                leftSum = sum;
                maxLeft = i;
            }
        }

        int rightSum = Integer.MIN_VALUE;
        sum = 0;
        int maxRight = mid + 1;

        // Find the maximum sum on the right side
        for (int j = mid + 1; j <= high; j++) {
            sum = sum + A.get(j);
            if (sum > rightSum) {
                rightSum = sum;
                maxRight = j;
            }
        }

        // Return buy and sell points with maximum sum
        Vector<Integer> result = new Vector<Integer>();
        result.add(maxLeft); // Buy point
        result.add(maxRight); // Sell point
        return result;
    }
```

```java
    public static void main(String[] args) {
        // Example usage:
        StockTrader stockTrader = new StockTrader();

        // Create a sample vector representing stock prices
        Vector<Integer> stockPrices = new Vector<>();
        stockPrices.add(7);
        stockPrices.add(1);
        stockPrices.add(5);
        stockPrices.add(3);
        stockPrices.add(6);
        stockPrices.add(4);

        // Call FindMaximumSubarray method
        Vector<Integer> result = stockTrader.FindMaximumSubarray(stockPrices, 0,
stockPrices.size() - 1);

        // Display the result
        System.out.println("Optimal Buy and Sell Points: [" + result.get(0) + ",
" + result.get(1) + "]");
    }
}
```



ALL CODE WORKS AS INTENDED !!