A paper with text and images

Description automatically generated

A white paper with black text and numbers

Description automatically generated

A white paper with black text

Description automatically generated

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) + "]");

        }

    }

A screenshot of a computer

Description automatically generated

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;

    }

    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) + "]");

    }

}

A screenshot of a computer program

Description automatically generated

ALL CODE WORKS AS INTENDED !!