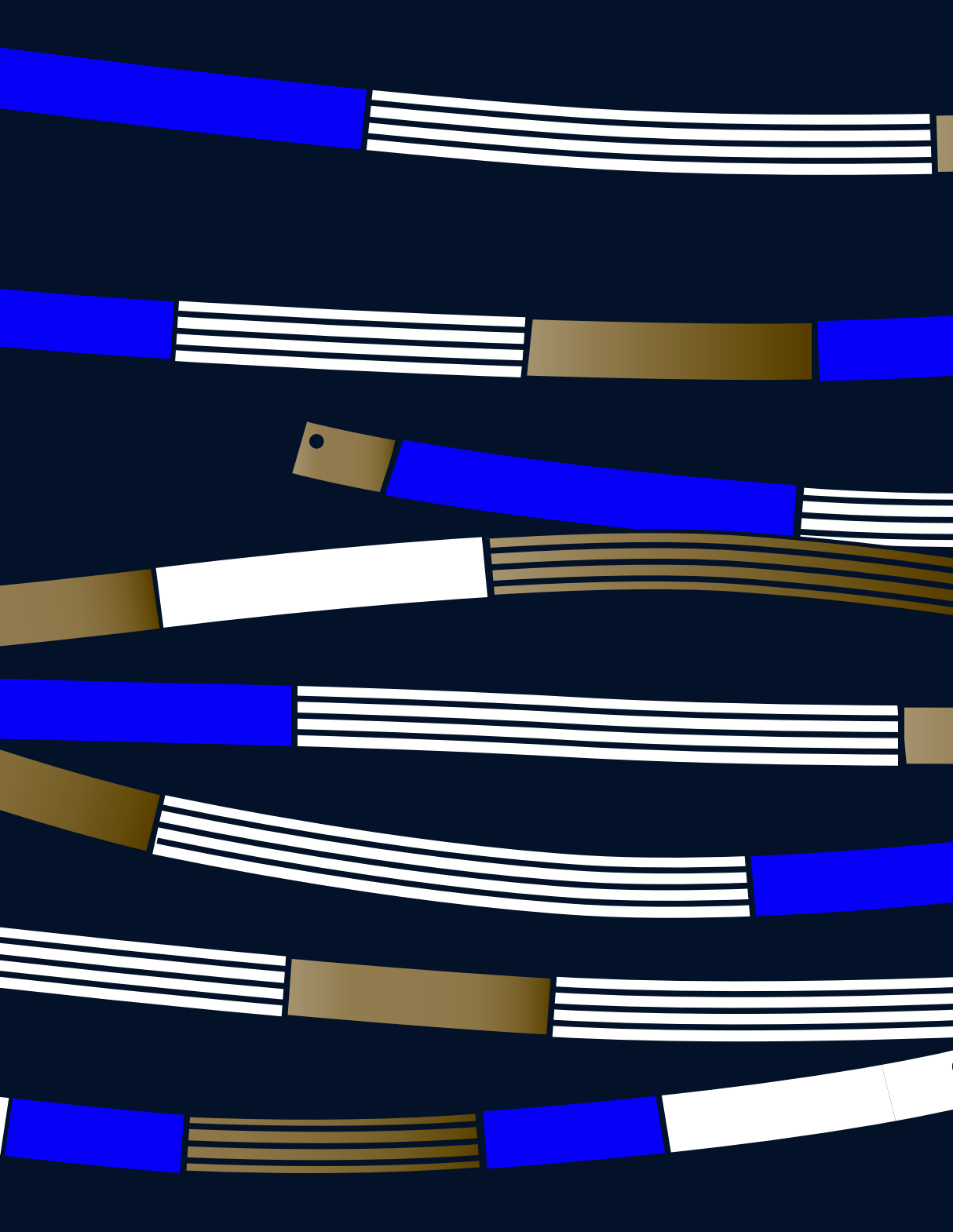


① maximum Sub array Sum (Brute force)

→ Here we need two for loop.

Outer loop is the starting index of Subarray

Inner loop for traversing the subarray every time and Counting the Sum.

* Here we need two Variable, i.e - CurrentSum, maxSum
  * Current Sum help us to Calculate the Current Sum of Subarray
  * And maxSum will store the maximum Sum of each Sub array

So we always Compared maximumSum with Current Sum

Code

```
// function definition {
int maxSum = Integer.MIN_VALUE;
int CurrentSum = 0;

for (int i=0; i < nums.length; i++) {
    for (int j = i; j < nums.length; j++) {
        for (int k = i; k < j; k++) {
        }
        CurrentSum += nums[k];
    if (maxSum < CurrentSum) {
```

```
            maxSum = currentSum;
        }
        currentSum = 0;
    }
}
System.out.println("max SubArray Sum :" + maxSum);
}
```

(2) maximum sum subarray (Prefix Sum)

Let's Say We have Array

{1, 9, 7, -1, -2} then PrefixArray should be

{1, 10, 17, 16, 14}

(+9) (+7) (+9) (+(-2))

We can achieve this by traverse the original Array

If We look at the Element of Prefix Array,

```
PrefixArray[i] = PrefixArray[i-1] + array[i];
```

After that instead of 3 for loop, We have to declare 2 for loop which time Complexity = $O(n^2) < O(n^3)$

V.V. Important *

```
And the SubArray Sum Can be found by this
currentSum = start == 0 ? prefix[end] :
                          prefix[end] - prefix[start-1];
```

Code

```
//function definition {
int[] prefix = new int [array.length];
int CurrentSum =0 ; maxSum = Integer.MIN_VALUE;
Prefix [0] = array[0];
for (int i=1 ; i< array.length; i++) {
    Prefix [i] = Prefix [i-1] + array [i];
}

for (int i=0 ; i < array.length; i++) {
    for (int j= i ; j < array.length ; j++) {
        Current Sum = i ==0 ? Prefix [j] :
                    Prefix [end] - Prefix [start-1];
        if (maxSum < CurrentSum) {
            maxSum = CurrentSum;
        }
    }
}
System.out.println ("Max Sum SubArray :" + maxSum);
}
```

③ Max Sub Array Sum. (Kadane's Algorithm)

+ve +ve = +ve
+ve + -ve = +ve
+ve + -ve = -ve

This Algorithm Says that,

We Initialize two Variable i.e

* CurrentSum = 0
* and maxSum = $-\infty$
* and one for loop that traverse the Array only one time
* and We Calculate CurrentSum, After Added the Element to the CurrentSum, We Should Check that if Sum is Still less than Zero, then assign Zero to CurrentSum instead of negative number.
* When the CurrentSum is +ve (positive)
  * Compare maxSum with CurrentSum and Store maximum Value to maxSum
* At last return maxSum;

* This Algorithm not gonna Work, if all the Element in the Array are negative (-ve)
* To Work this Algorithm on an Array, there Should be atleast one +ve Element in the Array

Q> Trapping Rain Water

Here we need 2 max Variable i.e

```
int leftmax = Integer.MIN_VALUE;
int Rightmax = Integer.MIN_VALUE;
```

and two Array that Stored the max Value from left side and right side

```
int [] leftmaxA = new int [array.length]
int [] rightmaxA = new int [array.length]
```

→ In leftmaxA we will assign the maximum bar Value as per the iteration from left to right

Similarly, In rightmaxB we will assign the Maximum bar value as per the iteration from right to left.

After Stored both the Array.

Again we need to iterate over both the Array at a time to find out minimum value / minimum bar height

and Subtract the bar height by the respective index of original bar height contain Array.

and Add the value After Subtraction to a Variable

and return that Variable.

☆ Trapping Rain Water

Code for main Water trapped

```
// function definition{ int WaterTrapped = 0;
int leftMax = Integer.MIN_VALUE;
int RightMax = Integer.MIN_VALUE;

int[] leftMaxArray = new int[array.length];
int[] RightMaxArray = new int[array.length];

for (int i = 0; i < array.length; i++) {
        if (leftMax < array[i]) {
            leftMax = array[i];
        }
    leftMaxArray[i] = leftMax;
}

for (int j = array.length-1; j >= 0; j--) {
        if (RightMax < array[j]) {
            rightMax = array[j];
        }
    RightmaxArray[j] = rightmax;
}

for (int K = 0; K < array.length; K++) {
    int minimum = Math.min(leftMaxArray[K],
                           RightmaxArray[K]);
    WaterTrapped += minimum - array[K];
}

System.out.println("Max Water can trapped :" + WaterTrapped);
}
```

☆ Best time to Buy and Sell Stock

To findout the best time to buy stock, it should be minimum price at all.

- So At first, we have to findout the minimum price of the array that gives the stock price.

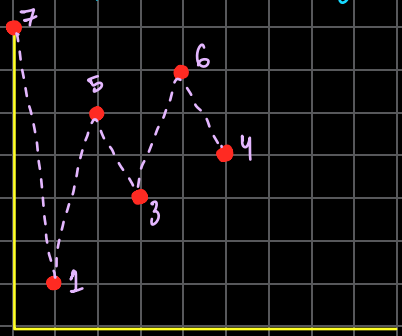
⇒ After found the minimum stock price, We should return the index of that minimum stock.

- Then we have to iterate over the Array from the minimum stock price index to end, to findout the maximum price in which we can sell our stock.

- After found the maximum stock price we're subtract the maximum stock Price Value by the minimum price.

and return the Profit.

Best time to buy and Sell the stocks : { 7, 1, 5, 3, 6, 4 }



index 0 = 7

No Number greater than 7 in {1,5,3,6,4}

= (0)

index 1 = 1

maximum Number After 1 = 6

Profit = 6 - 1 = (5)

index 2 = 5

maximum Number After 5 = 6

Profit = 6 - 5 = (1)

index 3 = 3

maximum Number After 3 = 6

Profit = 6 - 3 = (3)

index 4 = 6

max Number After 6 = Null

Profit = 0

index 5 = 4

Profit = 0