Prefix Sum + Sliding Window [Day 6]

08 September 2024 09:24

Prefix Sum + Sliding Windows [Day 6]

Problem Statement:

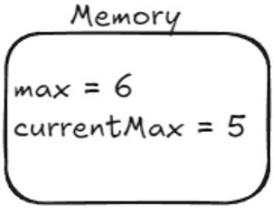
Given an array of integers, find the maximum sum of any contiguous subarray.

Example:

- **Input:**
  - `arr[] = {-2, 1, -3, 4, -1, 2, 1, -5, 4}`
- **Output:**
  - `6` (because the contiguous subarray `[4, -1, 2, 1]` has the maximum sum `6`)


```
max = arr[0];
currentmax = arr[0];
loop through i = 1 ;  i < n -1;
currentmax = Max(arr[i] , currentSum + arr[i]);
max = Max(max , currentmax);
return max;
```

[Kadane's Algorithm]



cur = max(4, 1+4) = 5

return max;

Kadane's Algorithm

```java
import java.util.Scanner;
public class MaximumSumOfSubArray {
    public static int findMaxSubarraySum(int[] arr){
        int maxSum = arr[0];
        int currentSum = arr[0];

        for(int i = 1; i< arr.length; i++){
            currentSum = Math.max(arr[i], currentSum + arr[i]);
            maxSum = Math.max(maxSum , currentSum);
        }
```

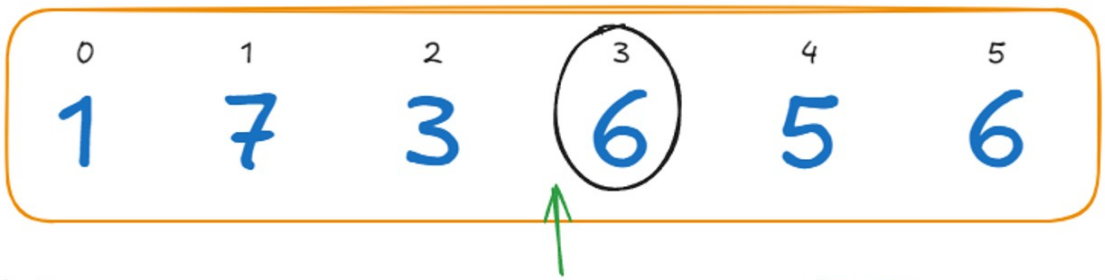
```
            currentSum = Math.max(arr[i], currentSum + arr[i]);
            maxSum = Math.max(maxSum , currentSum);
        }
        return maxSum;
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        System.out.println("Enter the number of Elements  in  the array : ");

        int n = scn.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of an array : ");
        for(int i = 0 ; i < n ; i++){
            arr[i] = scn.nextInt();
        }
        int maxSum = findMaxSubarraySum(arr);
        System.out.println(maxSum);
    }
}
```

Problem Statement:

Given an array `arr[]`, find the pivot index of the array. A pivot index is defined as the index where the sum of all the elements to its left is equal to the sum of all the elements to its right. If no such index exists, return `-1`.

Example:

- **Input:**
  - `arr[] = [1, 7, 3, 6, 5, 6]`
- **Output:**
  - `3` (because the sum of elements to the left of index `3` is `1 + 7 + 3 = 11`, and the sum of elements to the right is `5 + 6 = 11`)



leftSum = 0;
totalSum = 0;

Memory

```
leftSum = 0;
totalSum = 0;

Loop 0 to n-1
    totalSum += arr[i];

loop 0 to n-1
      rightSum = = totalSum - leftSum - arr[i];
      if(leftSum == rightSum ) -> return i;
        leftSum += arr[i]

return =1
```

Memory

```
leftSum = 11
TotalSum = 28;

rightSum = 11
```

```java
import java.util.Scanner;
public class PivotIndex {
    public static int findPivotIndex(int[] arr){
        int totalSum = 0;
        int leftSum = 0;

        for(int num : arr){
            totalSum += num;
        }

        for(int i=0 ; i<arr.length ; i++){
            int rightSum = totalSum - leftSum - arr[i];

            if(leftSum == rightSum){
                return i; // pivot Index or Equilibrium index
            }
            leftSum +=arr[i];
        }
        return -1; // No pivot index found
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        System.out.println("Enter the number of Elements  in  the array : ");

        int n = scn.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of an array : ");
        for(int i = 0 ; i < n ; i++){
            arr[i] = scn.nextInt();
        }
        int pivotIndex = findPivotIndex(arr);

        if(pivotIndex != -1){
            System.out.println("The pivot Index is " + pivotIndex);
```

```
        if(pivotIndex != -1){
            System.out.println("The pivot Index is " + pivotIndex);
        }else{
            System.out.println("No Pivot Index Found.");
        }
    }
}
```

-1 3 7 2 3 4 7

Problem: Buy and Sell Stock

Problem Statement:

Given an array where the `i-th` element represents the price of a stock on day `i`, you want to maximize your profit by choosing a single day to buy and a different day in the future to sell. You need to find the maximum profit you can achieve from a single transaction.

Example:

- **Input:**
  - `prices[] = [7, 1, 5, 3, 6, 4]`
- **Output:**
  - `5` (buy on day `2` (price = 1) and sell on day `5` (price = 6), profit = 6 - 1 = 5)

```
int minPrice = Infinity;
int maxProfit = 0;

for(price: prices){
    minPrice = min(minPrice , price)'
    int profit = price - minPrice.
    maxProfit = Max(maxProfit  ,profit)

retrun maxProfit
```

Memory

minPrice = 1
price[5] = 4
maxProfit =5
profit = 3

```
import java.util.Scanner;
public class BuyAndSellStocks {

    public static int maxProfit(int[] prices){
        int minPrice = Integer.MAX_VALUE;
        int maxProfit = 0;
        for(int price : prices){
            minPrice = Math.min(minPrice, price);
```

```
        int maxProfit = 0;
        for(int price : prices){
            minPrice = Math.min(minPrice, price);
            int profit = price - minPrice;
            maxProfit = Math.max(maxProfit, profit);
        }
        return maxProfit;
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        System.out.println("Enter the number of Days: : ");
        int n = scn.nextInt();
        int[] prices = new int[n];
        System.out.println("Enter the stocks
                            prices for each day : ");
        for(int i = 0 ; i < n ; i++){
            prices[i] = scn.nextInt();
        }
        int profit = maxProfit(prices);
        System.out.println("The max Profit is : " + profit);
    }
}
```

Problem: Running Sum of an Array

Problem Statement:

Given an array `nums`, you need to compute the running sum of the array. The running sum of an array is a new array where the value at each index `i` is the sum of all elements from the start of the array up to index `i`.

Example:

- **Input:**
  - `nums[] = [1, 2, 3, 4]`
- **Output:**
  - `[1, 3, 6, 10]`
  - Explanation:
    - Running sum at index `0` is `1`
    - Running sum at index `1` is `1 + 2 = 3`
    - Running sum at index `2` is `1 + 2 + 3 = 6`
    - Running sum at index `3` is `1 + 2 + 3 + 4 = 10`

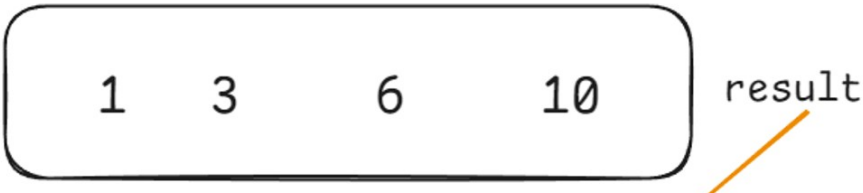
arr

runningSum

runningSum[i] = runningSum[i-1] + arr[i]

```
runningSum[i] = runningSum[i-1] + arr[i];
```

```java
import java.util.Scanner;
import java.util.Arrays;
public class RunningSum {
    public static int[] runningSum(int[] nums){
        int n = nums.length;
        int[] runningSum = new int[n];

        if(n > 0){
            runningSum[0] =  nums[0];
            for(int i =1 ; i<n ; i++){
                runningSum[i] = runningSum[i-1] + nums[i];
            }
        }
        return runningSum;
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        System.out.println("Enter the number of Elements  in  the array : ");
        int n = scn.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the elements of an array : ");
        for(int i = 0 ; i < n ; i++){
            nums[i] = scn.nextInt();
        }
        int[] result = runningSum(nums);
        System.out.println("The running sum of an array : " +
        Arrays.toString(result));
    }
}
```



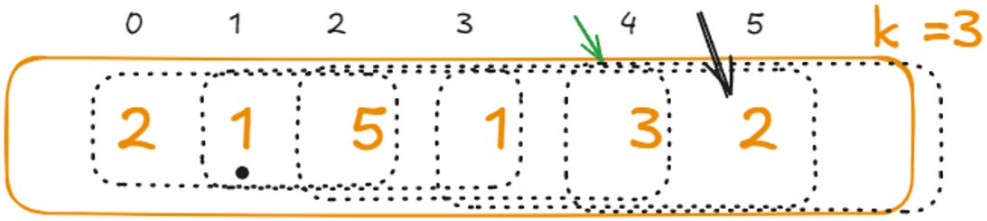
Instead of iteration to print all the elements of result array . We use a new technique called

Instead of iteration to print all the elements of result array . We use a new technique called as Arrays.toString(result) to print all the element of Result array.

"Sliding Window Technique"

Find the maximum sum of `k` consecutive elements in an array.

- **Input:** `arr[] = [2, 1, 5, 1, 3, 2]`, `k = 3`
- **Output:** `9` (because the maximum sum of 3 consecutive elements is `5 + 1 + 3 = 9`)



max = 9

```
int  n = arr.length;
if(n < k){
    return -1;
}
int maxSum = 0;
for(int i =0; i< k; i++){
    maxSum += arr[i];
}
int windowSum = maxSum;
for(int i = k ; i < n; i++){
    windowSum += arr[i] - arr[i-k];
    maxSum = Math.max(maxSum, windo
}
return maxSum;
```

Memory

n= 6
k = 3
maxSum =9
windowSum = 6

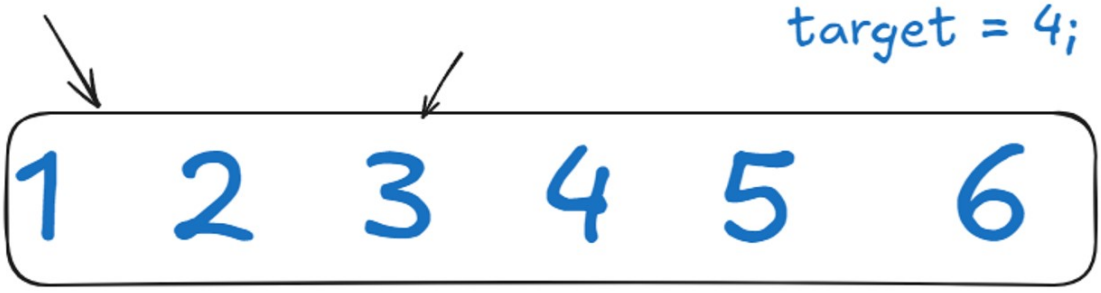
maxsum = 14
windowSum =-5

windowSum =-5

2 Pointer Approach

Find if there exist two elements in a sorted array whose sum equals a given target.

- Input: `arr[] = [1, 2, 3, 4, 6]`, `target = 8`
- Output: `True` (because `2 + 6 = 8`)

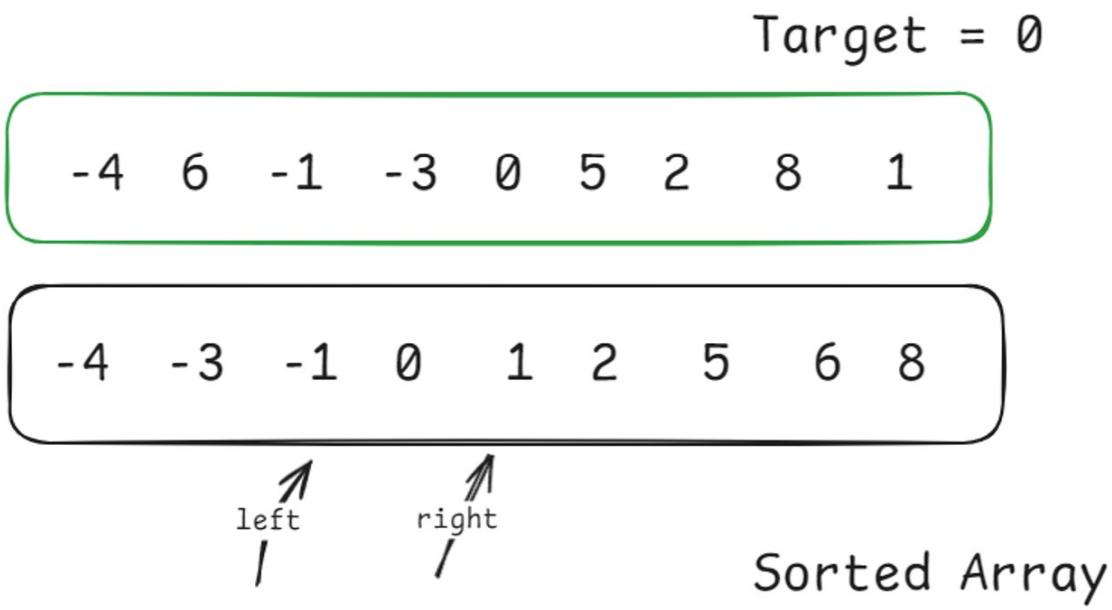


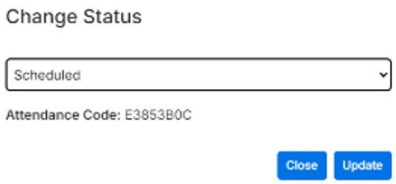

```
public class TwoPointerExample {

    public static boolean hasPairEqualToTarget(int[] arr , int target){
        int left = 0;
        int right = arr.length - 1;

        while(left < right){
            int sum = arr[left] + arr[right];
            if(sum == target){
                return true;
            }else if(sum < target){
                left++;
            }else{
                right--;
            }
        }
        return false;
    }
    public static void main(String[] args) {
        int[] arr = {1,2,3,4,5,6};
        int target = 8;
        boolean result = hasPairEqualToTarget(arr , target);
        System.out.println(result);
    }
}
```

- Sort the array if its not sorted.

}

- Sort the array if its not sorted.
  usings Arrays.Sort().





"Two Sum"

Problem Statement:

Given a sorted array of integers `arr[]` and a target integer `target`, find the indices of two numbers such that they add up to the target.

Example:

- Input: `arr[] = [2, 7, 11, 15]`, `target = 9`
- Output: `[0, 1]` (because `arr[0] + arr[1] = 2 + 7 = 9`)

```
public class TwoSum {

    public static int[] twoSum(int[] arr , int target){
        int left = 0;
        int right = arr.length - 1;

        while(left < right){
```

```
        int right = arr.length - 1;

        while(left < right){
            int sum = arr[left] + arr[right];
            if(sum == target){
                return new int[] {left , right};
            }else if(sum < target){
                left++;
            }else{
                right--;
            }
        }
        return new int[] {};
    }
    public static void main(String[] args) {
        int[] arr = {2 , 7 , 11 ,15};
        int target = 9;
        int[] result = twoSum(arr , target);

        if(result.length > 0){
            System.out.println("Indices of the 2 numbers are : "+
            result[0] + " and " + result[1]);
        }else{
            System.out.println("No 2Sum solution found");
        }
    }
}
```