**Kadane’s Algorithm:**

Kadane’s algorithm is used to calculate the maximum sub array sum problem with a complexity of O(n).

Here we calculate the sum of consecutive elements and compare with maximum sum if the sum is greater than max sum we update the value of max sum.

If the sum becomes negative then we update the max sum if needed, then we reset the value of sum to 0. So that it won’t affect the sum of the coming sub array.

Because negative values will however decrease the value, but we need the maximum value.

**Program:**

package com.mycompany.app.DataStructures;

import java.util.Scanner;

public class LargerstSubArraySum{

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

int sum = 0, max = Integer.MIN\_VALUE;

for(int i=0;i<n;i++) {

arr[i] = sc.nextInt();

}

for(int i=0;i<n;i++) {

sum+=arr[i];

if(max<sum)

max = sum;

if(sum<0) {

if(max<sum)

max = sum;

sum = 0;

}

}

if(max<sum)

max = sum;

System.out.println("Maximum Sub Array sum is: "+max);

}

}

We can also retrieve the indices of the starting and the ending indices of the Sub array that give the maximum sum.

**Program**

package com.mycompany.app.DataStructures;

import java.util.Scanner;

public class MostWater{

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

for(int i=0;i<n;i++) {

arr[i] = sc.nextInt();

}

int[] leftMax = new int[n];

leftMax[0] = arr[0];

for(int i=1;i<n;i++) {

int max = Math.max(leftMax[i-1],arr[i]);

leftMax[i] = max;

}

int[] rightMax = new int[n];

rightMax[n-1] = arr[n-1];

for(int i=1;i<n;i++) {

int max = Math.max(rightMax[n-i],arr[n-i-1]);

rightMax[n-i-1] = max;

}

int sum = 0, start = 0, end = n-1, max = Integer.MIN\_VALUE,x = 0,y=0;

for(int i=0;i<n;i++) {

int min = Math.min(leftMax[i],rightMax[i]);

sum+=(min-arr[i]);

if(min-arr[i]==0) {

if(max<sum) {

max = sum;

start = x;

end = i;

}

sum = 0;

x=i;

}

}

if(max<sum) {

max = sum;

start = x;

end = n-1;

}

System.out.println("The subArray from indices "+start+" to "+end+" gives the max sum of "+max);

}

}

**Program to calculate the amount of rain water trapped:**

package com.mycompany.app.DataStructures;

import java.util.Scanner;

public class RainWaterTrap{

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

int[] rightMax = new int[n];

int[] leftMax = new int[n];

for(int i=0;i<n;i++) {

arr[i] = sc.nextInt();

}

leftMax[0] = arr[0];

rightMax[n-1] = arr[n-1];

for(int i=1;i<n;i++) {

int max = Math.max(leftMax[i-1],arr[i]);

leftMax[i] = max;

}

for(int i=1;i<n;i++) {

int max = Math.max(arr[n-i-1],rightMax[n-i]);

rightMax[n-i-1] = max;

}

int sum = 0;

for(int i=0;i<n;i++) {

int max = Math.min(rightMax[i],leftMax[i]);

sum = sum+(max-arr[i]);

}

System.out.println(sum);

}

}

**============== SORTING ALGORITHMS ===================**

**Bubble sort:**

In Bubble sort, we will have two loops.

The i th element in the first loop is compared to all the elements in the second loop.

If the element in the second loop is lesser than the element of the first loop, we swap the elements.

In this way, the largest element gets to the ending index first.

By the end program we get a sorted array.

The time complexity is O(n^2).

The space complexity is O(1).

**Program:**

package com.mycompany.app.DataStructures;

import java.util.Scanner;

public class BubbleSort{

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

for(int i=0;i<n;i++) {

arr[i] = sc.nextInt();

}

for(int i=0;i<n-1;i++) {

for(int j = i+1;j<n;j++) {

if(arr[i]>arr[j]) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

for(int i=0;i<n;i++) {

System.out.print(arr[i]+" ");

}

}

}

**Selection Sort:**

In selection sort we have two loops.

In the first loop we maintain the index of the minimum element.

We will first consider the i th index to be the minimum value index and then compare the value at that index to all the values in the second loop.

If there is a minimum value than the stored index value’s value, we update the minimum index.

At the end, we swap the values of the min index with the i th index.

In this way first the minimum element will be sorted in the first place and we will achieve the sorted array by sorting all the minimum elements.

The time complexity is O(n^2).

The space complexity is O(1).

**Program**

package com.mycompany.app.DataStructures;

import java.util.Scanner;

public class SelectionSort{

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

for(int i=0;i<n;i++) {

arr[i] = sc.nextInt();

}

for(int i=0;i<n-1;i++) {

int min = i;

for(int j=i+1;j<n;j++) {

if(arr[min]>arr[j]) {

min = j;

}

}

int temp = arr[min];

arr[min] = arr[i];

arr[i] = temp;

}

for(int i=0;i<n;i++) {

System.out.print(arr[i]+" ");

}

}

}

**Insertion sort:**

In Insertion sort we have two loops.

The first loop runs from index number 1. We store the element value at

i th index. And we will also have j th value to run the second loop.

In the second loop we check if the j th value is correct and and also checks if the j th value element is lesser than the i th value element. If lesser we continue to next iteration of the first loop, if not we continue until we achieve that.

In the second loop during iteration the j th index elements are brought by swapping them with each other.

Time complexity is O(n^2).

Space Complexity is O(1).

**Program**

package com.mycompany.app.DataStructures;

import java.util.Scanner;

public class InsertionSort{

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

for(int i=0;i<n;i++) {

arr[i] = sc.nextInt();

}

for(int i=1;i<n;i++) {

int key = arr[i];

int j = i-1;

while(j>=0 && arr[j]>key) {

arr[j+1] = arr[j];

j--;

}

arr[j+1] = key;

}

for(int i=0;i<n;i++) {

System.out.print(arr[i]+" ");

}

}

}