Problems on Array-4

Assignment Solutions

PW SKILLS

Q1. Given an array of integers of length n, and an integer m, (m < n), divide the array into 2 subarrays such that 1 part contains m elements and the other part contains the rest of the elements. This division has to be done such that the difference between the sum of elements of both the sub arrays is the maximum. You have to print the maximum difference in the sum possible.

Input:
N = 6
Arr[] = 7 4 6 0 8 2
M = 2
Expected Output:
23

Explanation:

- Sort the array
- Difference will be maximum if one group contains the smallest m elements and rest are in other group or 1 group contains m largest element and rest are in other group.
- Compare the 2 differences calculated and print the maximum difference.

Code:

```
import java.util.Scanner;
import java.util.Arrays;
public class Test {
   public static void main(String[] args) {
       Scanner scn = new Scanner(System.in);
       System.out. println("Enter the length of the array: ");
       int n = scn.nextInt();
       int[] arr = new int[n];
       for(int i = 0; i < n; i++){
           arr[i] = scn.nextInt();
       }
       int m = scn.nextInt();
       Arrays.sort(arr);//for max sum, we need to solve for extreme elements
       int lsum1 = 0; //calculate sum for both parts from left end with subarray of size m having min elements
       int rsum1 = 0;
       int i = 0;
       while(i <= m-1){
           lsum1 += arr[i];
           i++;
       }
       while(i < n){
           rsum1 += arr[i];
           i++;
       }
       int diff1 = Math.abs(lsum1 - rsum1); //diff when m size subarray contains min elements
       int lsum2 = 0;//calculate sum for both parts from right end with subarray of size m having max elements
       int rsum2 = 0;
       int j = n-1;
       while(j >= n - m){
           rsum2 += arr[j];
           j--;
       }
```

```
while(j >= 0){
            lsum2 += arr[j];
            j--;
        }
        int diff2 = Math.abs(lsum2 - rsum2);//diff when m size subarray contains min elements
        System.out.print(Math.max(diff1, diff2));
    }
}
```

```
6
7 4 6 0 8 2
2
23
Process finished with exit code 0
```

Q2. Given an integer array arr consisting of 0's and 1's only, return the max length of sequence which contains equal numbers of 0 and 1. If no such subarray exists, print -1.

Input:
N = 7
arr=[0,1,1,0,1,1,1]
Expected Output:
4

Explanation:

- Keep a pointer maxsize to track the largest subarray and traverse the array.
- We keep a pointer sum, everytime we encounter a 0, we add -1 to the sum, else we add 1.
- Is sum = 0, this will be only when we have equal number of 0s and 1s. At this point, keep updating the maxsize.

Code:

```
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        System.out. println("Enter the length of array: ");
        int n = scn.nextInt();
        int[] arr = new int[n];
        for(int i = 0; i < n; i++){
            arr[i] = scn.nextInt();
        }
        int sum = 0;
        int maxsize = -1; //initialize from -1 because incase no subarray found, we are still printing maxsize in
        the end which will then be -1 only
        for (int i = 0; i < n - 1; i++) {
            sum = (arr[i] == 0) ? -1 : 1; //-1 indicates presence of 0 and 1 indicates presence of 1 so when equal
            number of 0 and 1 are present, sum = 0
             for (int j = i + 1; j < n; j++) {
                if (arr[j] == 0)
                    sum += -1;
                else
```

```
                sum += 1;
                if (sum == 0 && maxsize < j - i + 1) {
                    maxsize = j - i + 1;
                }
            }
        }
        System.out.println(maxsize);
    }
}
```

```
7
0 1 1 0 1 1 1
4

Process finished with exit code 0
```

Q3. There is a biker going on a road trip. The road trip consists of n + 1 points at different altitudes. The biker starts his trip on point 0 with altitude equal 0.
You are given an integer array gain of length n where gain[i] is the net gain in altitude between points i and i + 1 for all (0 <= i < n). Return the highest altitude of a point.

Input:
n = 5
gain = [-5,1,5,0,-7]

Expected Output:
1

Explanation:

- Create a blank array of n+1 size.
- The altitude of point 0 is 0, so ans[0] = 0
- For the further points, altitude = altitude of previous point + current gain.
- Calculate the max altitude of the array.

Code:

```
import java.util.Scanner;
public class Test{
   public static void main(String[] args){
       Scanner scn = new Scanner(System.in);
       System.out.println("Enter the length of array");
       int n = scn.nextInt();
       int[] gain = new int[n];
       System.out.println("Enter the elements of array");
       for(int i = 0; i < n; i++){
           gain[i] = scn.nextInt();
       }
       int[] ans = new int[n+1];
       ans[0] = 0;
       for(int i = 1; i < n+1; i++){
           ans[i] = ans[i-1] + gain[i-1];
       }
       int max = Integer.MIN_VALUE;
       for(int i = 0; i < n+1; i++){
           max = Math.max(max, ans[i]);
       }
       System.out.println(max);
   }
}
```

```
/Library/Java/JavaVirtualMachines/jdk-
Enter the length of array
5
Enter the elements of array
-5 1 5 0 -7
1

Process finished with exit code 0
```

Q4. Given a 0-indexed integer array nums, find the leftmost middleIndex (i.e., the smallest amongst all the possible ones).
A middleIndex is an index where nums[0] + nums[1] + ... + nums[middleIndex-1] == nums[middleIndex+1] + nums[middleIndex+2] + ... + nums[nums.length-1].
If middleIndex == 0, the left side sum is considered to be 0. Similarly, if middleIndex == nums.length - 1, the right side sum is considered to be 0.
Return the leftmost middleIndex that satisfies the condition, or -1 if there is no such index.

Input:
n = 5
nums = [2,3,-1,8,4]

Expected Output:
3

Explanation:

- Calculate prefix sum from left and right both sides and store it in 2 separate arrays.
- The prefix sum for ith index should not include the ith element.
- Traverse the arrays and return the index where the prefix sum for both the left and right arrays is the same.
- Return -1 in the end, which will run if we find no middle index.

Code:

```
import java.util.Scanner;
public class Test{
   public static void main(String[] args){
       Scanner scn = new Scanner(System.in);
       System.out.println("Enter the length of array");
       int n = scn.nextInt();
       int[] nums = new int[n];
       System.out.println("Enter the elements of array");
       for(int i = 0; i < n; i++){
           nums[i] = scn.nextInt();
       }
       int[] left = new int[n];
       left[0] = 0;
       int[] right = new int[n];
       right[n-1] = 0;
       for(int i = 1; i < n; i++){
           left[i] = left[i-1] + nums[i-1];
       }
       for(int i = n-2; i >= 0; i--){
           right[i] = right[i+1] + nums[i+1];
       }
       for(int i = 0; i < n; i++){
           if(left[i] == right[i]){
               System.out.println(i);
               return;
           }
       }
       System.out.println(-1);
   }
}
```

```
/Library/Java/JavaVirtualMachines/jdk-
Enter the length of array
5
Enter the elements of array
2 3 -1 8 4
3

Process finished with exit code 0
```