Problems on Array - 3

Assignment Solutions

SKILLS

Q1. Given an integer m, n, and n integers, return true if the number of unique integers among the n integers is greater than or equal to m, else return false.(Integers appearing multiple times are all considered as 1 unique integer)

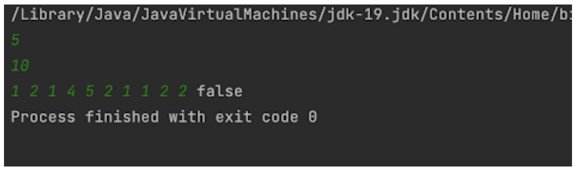
Input:
5
10
1 2 1 4 5 2 1 1 2 2
Expected Output:
false

Explanation:

- Store the integers in an array and sort the array
- Sorting will align all duplicates together
- Keep an index pointer initialized with 0
- Increment count for the element, now check uptill what index is the element getting repeated, and jump to the last index of its occurrence using a while loop.
- Increment index by 1 more so you get the next new element and increment count whenever you are out of your inner while loop

Code:

```
import java.util.Scanner;
import java.util.Arrays;
public class Test {
   public static void main(String[] args) {
       Scanner scn = new Scanner(System.in);
       int m = scn.nextInt();
       int n = scn.nextInt();
       int[] arr = new int[n]; //store the n integers in an array
       for(int i = 0; i < n; i++){
           arr[i] = scn.nextInt();
       }
       Arrays.sort(arr);//on sorting, same integers will get aligned in consecutive indices
       int i = 0;
       int count = 0;
       while(i < n){
           count++;
           while(i < n-1 && arr[i+1] == arr[i]){//skip duplicates of element
               i++;
           }
           i++; //increment 1 more as i was still pointing to the last duplicate of previous element
       }
       if(count >= m){
           System.out.print(true);
       }else{
           System.out.print(false);
       }
   }
}
```



```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/b
5
10
1 2 1 4 5 2 1 1 2 2 false
Process finished with exit code 0
```

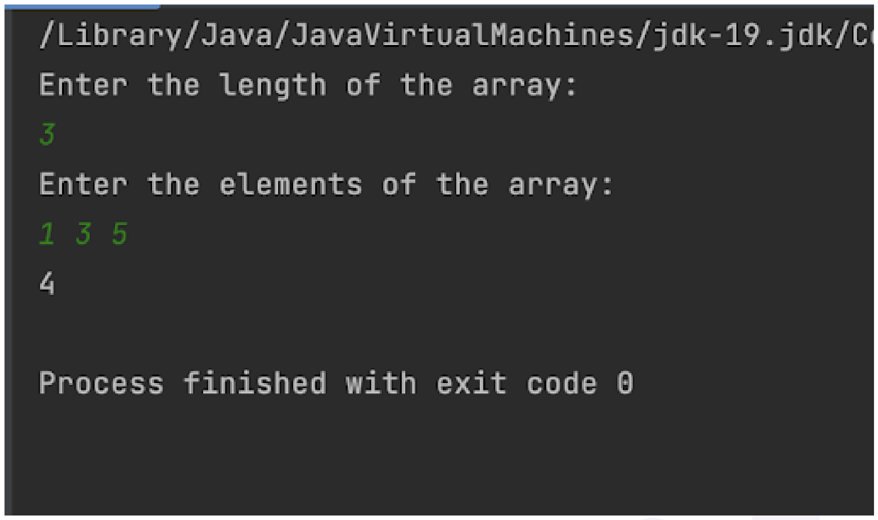
Q2. Given an integer array arr, return the number of consecutive sequences(subarrays) with odd sum.

Input 1:
N = 3
[1,3,5]
Expected Output:
4
Explanation:

- Odd + odd gives even sum, even + odd gives odd sum, even + even gives even sum
- If we know the number of even and odd subarrays that end at the previous element, we can figure out how many even and odd subarrays we have for element n.
- If n is even, we increase the number of even subarrays; the number of odd subarrays does not change.
- If n is odd, the number of odd subarrays is the previous number of even subarrays + 1. The number of even subarrays is the previous number of odd subarrays.

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 the array: ");
       int n = scn.nextInt();
       int[] arr = new int[n];
       System.out. println("Enter the elements of the array: ");
       for(int i = 0; i < n; i++){
           arr[i] = scn.nextInt();
       }
       int odd = 0, even = 0, sum = 0;
       for (int num : arr) {
           if (num % 2 == 1) {
               int temp = odd; //swap odd and even
               odd = even;
               even = temp;
               odd++;
           }
           else{
               even++;
           }
           sum += odd;
       }
       System.out.println(sum);
   }
}
```



```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/C
Enter the length of the array:
3
Enter the elements of the array:
1 3 5
4

Process finished with exit code 0
```

Q3. You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).
Find two lines that together with the x-axis form a container, such that the container contains the most water. Return the maximum amount of water a container can store.

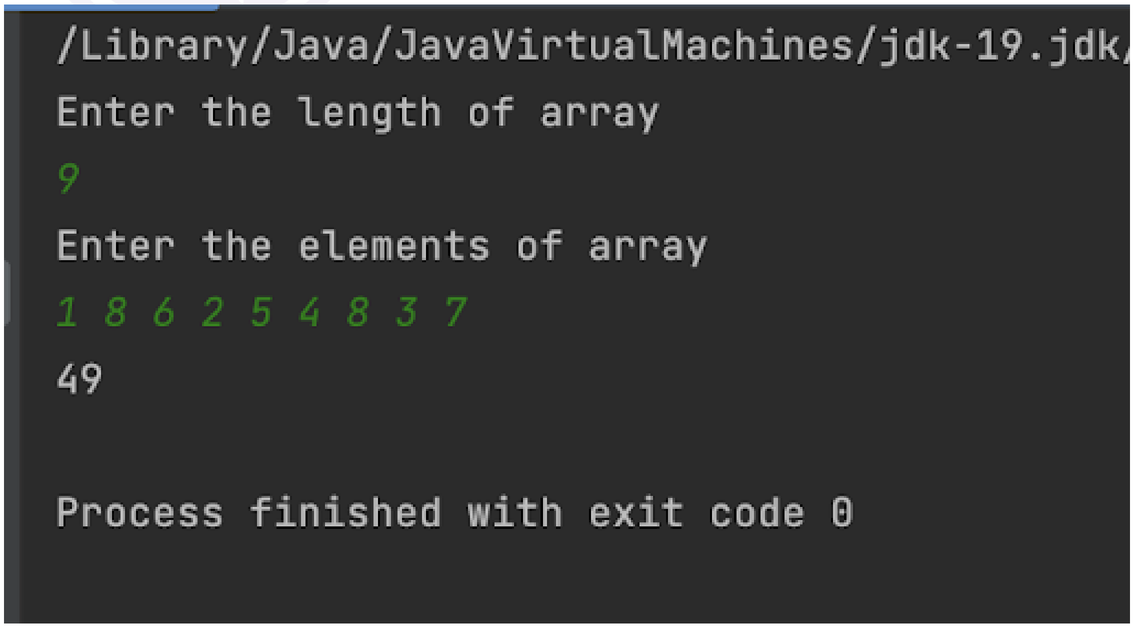
Input:
n = 9
height = [1,8,6,2,5,4,8,3,7]
Expected Output:
49

Explanation:

- Use 2 pointers, 1 from start and other from end of array
- Run a while loop till i is less than j, calculate width between the 2 bars as j-i and height will be the maximum of both the bars.
- Calculate the area everytime and keep the max
- Move the pointer whose height is less.

Code:

```java
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[] height = new int[n];
       System.out.println("Enter the elements of array");
       for(int i = 0; i < n; i++){
           height[i] = scn.nextInt();
       }
       int i = 0;
       int j = n-1;
       int ans = 0;
       while(i < j){
           int width = j-i;
           int ht = Math.min(height[i], height[j]);
           int area = ht * width;
           ans = Math.max(ans, area);
           if(height[i] < height[j]){
               i++;
           }else{
               j--;
           }
       }
       System.out.println(ans);
   }
}
```



```
/Library/Java/JavaVirtualMachines/jdk-19.jdk
Enter the length of array
9
Enter the elements of array
1 8 6 2 5 4 8 3 7
49

Process finished with exit code 0
```

Q4. Given a 1-indexed array of integers numbers that is already sorted in non-decreasing order, find two numbers such that they add up to a specific target number.
Return the indices of the two numbers added by one. Return -1 if pair does not exist.

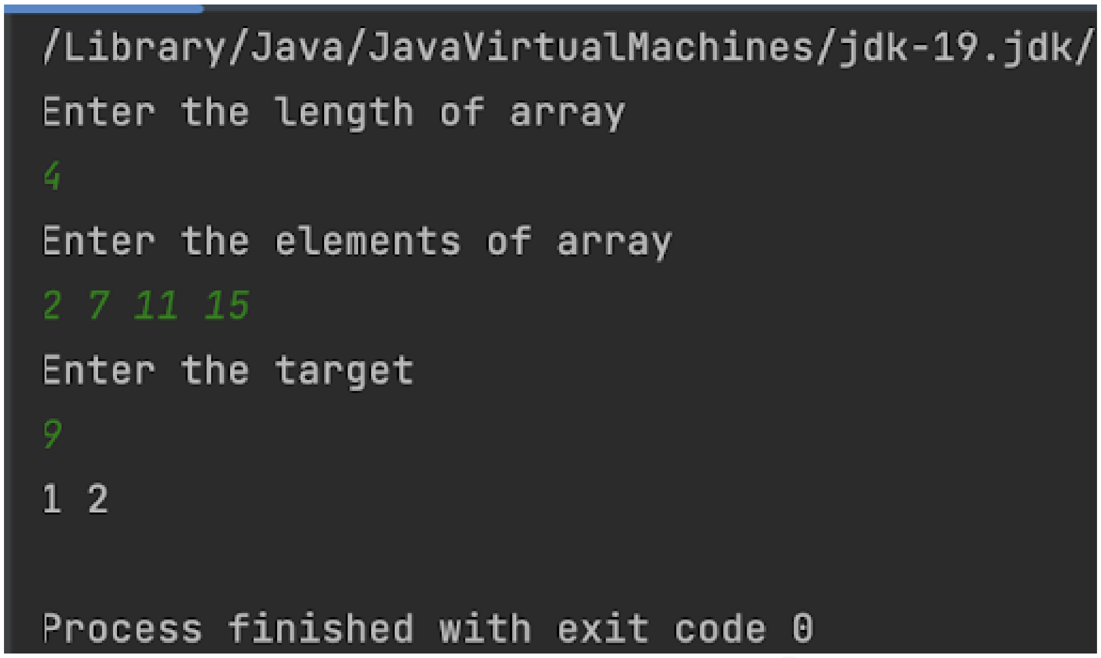
Input:
n = 4
numbers = [2,7,11,15]
target = 9
Expected Output:
1 2

Explanation:

- Use 2 pointers, 1 from start and other from end of array
- In a while loop for the condition, i < j, calculate the sum of elements at ith and jth index.
- If the sum meets target, print and return
- If sum exceeds the target, it means we need a smaller number, so decrement j.
- If sum is less than target, we need a bigger number, increment i.
- Print -1 in the end, this will run only when we haven't found any pair that adds upto the target.

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[] numbers = new int[n];
       System.out.println("Enter the elements of array");
       for(int i = 0; i < n; i++){
           numbers[i] = scn.nextInt();
       }
       System.out.println("Enter the target");
       int target = scn.nextInt();
       int i = 0;
       int j = n-1;
       while(i < j){
          if(numbers[i] + numbers[j] == target){
              System.out.println(++i + " " + ++j);
              return;
          }else if(numbers[i] + numbers[j] > target){
              j--;
          }else{
              i++;
          }
       }
       System.out.println(-1);
   }
}
```



```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/
Enter the length of array
4
Enter the elements of array
2 7 11 15
Enter the target
9
1 2

Process finished with exit code 0
```

Q5. Given an array sorted in increasing order, return an array of squares of each number sorted in increasing order

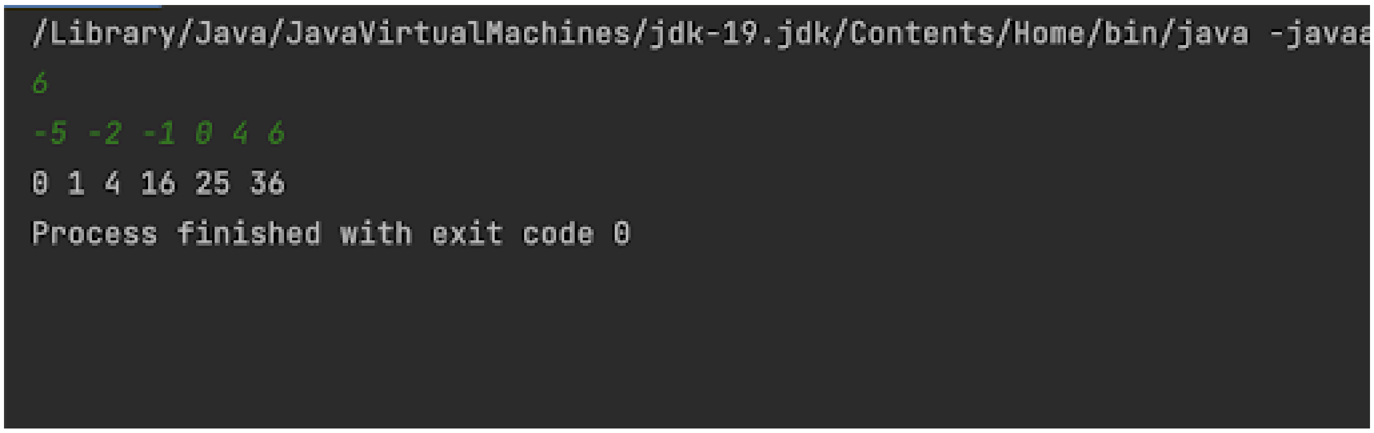
Input:
N = 6
Arr[] = [-5, -2, -1, 0, 4, 6]
Expected Output:
[0, 1, 4, 16, 25, 36]

Explanation:

- Using 2 pointer approach, first pointer will point to first negative element which will be at index 0, so no need to calculate.
- The second pointer will point to first positive element, for which we will traverse the array
- Create ans array(blank of size n), and keep track of its curr index using idx
- Calculate squares of both numbers at both neg and pos index and whichever is smaller, add it to ans array at idx and increment the pointers: idx and the one of which square of that element is added.
- If square of negative element is less, neg pointer is decremented as they are arranged in descending order relative to the array and in order of their greatness.
- If any of the pointers reach their index out of bounds, end the while loop, and add the remaining elements of the other pointer as it is after square.

Code:

```
import java.util.Scanner;
public class Test {
   public static void main(String[] args) {
       Scanner scn = new Scanner(System.in);
       System.out.print("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[] ans = new int[n];
       int idx = 0;
       int firstNonNegativeElementIndex = n;
       for(int i = 0; i < n; i++) {
           if(arr[i] >= 0) {
               firstNonNegativeElementIndex = i;
               break;
           }
       }
//using 2 pointers
       int negItr = firstNonNegativeElementIndex-1; //starting from largest amongst negative numbers
       int posItr = firstNonNegativeElementIndex; //starting from smallest amongst positive numbers
       while(negItr >= 0 && posItr < n) {
           int negElementSquare = arr[negItr]*arr[negItr];
           int posElementSquare = arr[posItr]*arr[posItr];
           if(negElementSquare < posElementSquare) {//whichever square is smallest, add to ans array first
               ans[idx++] = negElementSquare;
               negItr--;
           } else {
               ans[idx++] = posElementSquare;
               posItr++;
           }
       }
       while(negItr >= 0) {
           ans[idx++] = arr[negItr]*arr[negItr];
           negItr--;
       }
       while(posItr < n) {
           ans[idx++] = arr[posItr]*arr[posItr];
           posItr++;
       }
       for(int i = 0; i < n; i++){
           System.out.print(ans[i] + " ");
       }
   }
}
```



```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java -javaa
6
-5 -2 -1 0 4 6
0 1 4 16 25 36
Process finished with exit code 0
```