Problems on Array-2

Assignment Solutions

PW SKILLS

Q1. Given an array arr[] of size n, find the first repeating element. The element should occur more than once and the index of its first occurrence should be the smallest. If no repeating element exists, print -1. (Assume 1 based indexing)

Input:
n = 7
arr[] = {1, 5, 3, 4, 3, 5, 6}
Expected Output:
2

Explanation:

- Traverse the array starting from first index
- Check if the element repeats itself in the array using another inner loop.
- If element has been found in any of further indices, it is a repeating element, and since we are checking from index 0, it is also the first repeating element.
- Print the index and return(end the function right there).
- Output will be i+1, because question assumes 1 based indexing but in java it is always 0 based.
- In the end, print -1, we will reach this statement only if we don't find any repeating element.

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.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();
       }
       for(int i = 0; i < n; i++){
           for(int j = i+1; j < n; j++){
               if(arr[i] == arr[j]){ //first repeating element
                   System.out.print(i+1);
                   return;
               }
           }
       }
System.out.print(-1); //no repeating element found
}
}
```

```
7
1 5 3 4 3 5 6
2
Process finished with exit code 0
```

Q2. Given an array of positive and negative numbers, arrange them in an alternate fashion such that every positive number is followed by a negative and vice-versa maintaining the order of appearance. The number of positive and negative numbers need not be equal. Begin with a negative number. If there are more positive numbers, they appear at the end of the array. If there are more negative numbers, they too appear at the end of the array.

Input 1:
N = 6
arr[] = {1, 2, 3, -4, -1, 4}
Expected Output:
{-4, 1, -1, 2, 3, 4}

Explanation:

- We will maintain a variable to mark if the element is in its correct position or not. Let the variable be outofplace.
- Initially, it is -1.
- We will iterate over the array
- If outofplace is -1, we will check if the current index is out of place. If the current index is out of place we will update the outofplace with the index value.Else we will keep the value as it is.
- If the outofplace is not -1, we will search for the next index which has a different sign than that of the value that is present in outofplace position. Now we will pass these two positions to right rotate the array using our created function rightrotate.
- Rightrotate function will rotate the array elements to the left from cur index to outofplace index.
- Update the value of outofplace by 2 unit.

Code:

```
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 outofplace = -1; //An element is out of place if it is negative and at odd index (0-based index),
or if it is positive and at even index (0-based index).
       for (int index = 0; index < n; index++) {
           if (outofplace >= 0) {
               if (((arr[index] >= 0) && (arr[outofplace] < 0)) || ((arr[index] < 0) && (arr[outofplace] >= 0))) {
                   rightrotate(arr, n, outofplace, index);
                   if (index - outofplace >= 2)
                       outofplace = outofplace + 2;
                   else
                       outofplace = -1;
               }
           }
               if (outofplace == -1) { //if no entry has been flagged out-of-place
               if (((arr[index] >= 0)
                       && ((index & 0x01) == 0))
                       || ((arr[index] < 0)
                       && (index & 0x01) == 1))
                   outofplace = index;
           }
       }
       for(int i = 0; i < n; i++){
           System.out.print(arr[i] + " ");
       }
}
public static void rightrotate(int arr[], int n, int outofplace, int cur) {
       int tmp = arr[cur];
       for (int i = cur; i > outofplace; i--)
           arr[i] = arr[i - 1];
       arr[outofplace] = tmp;
   }
}
```

```
6
1 2 3 -4 -1 4
-4 1 -1 2 3 4
Process finished with exit code 0
```

Q3. Minimum Platforms
Given arrival and departure times of all trains that reach a railway station. Find the minimum number of platforms required for the railway station so that no train is kept waiting.

Consider that all the trains arrive on the same day and leave on the same day. Arrival and departure time can never be the same for a train but we can have arrival time of one train equal to departure time of the other. At any given instance of time, same platform can not be used for both departure of a train and arrival of another train. In such cases, we need different platforms.

Input 1:
n = 6
arr[] = {0900, 0940, 0950, 1100, 1500, 1800}
dep[] = {0910, 1200, 1120, 1130, 1900, 2000}
Expected Output:
3

Explanation:

- Sort the arrival and departure times of trains.
- Create two pointers i=1, and j=0, and a variable to store ans and current count plat
- Run a loop while i<n and j<n and compare the ith element of arrival array and jth element of departure array. If the arrival time is less than or equal to departure then one more platform is needed so increase the count, i.e., plat++ and increment i
- If the arrival time is less than or equal to departure then one more platform is needed so increase the count, i.e., plat++ and increment i
- Else if the arrival time is greater than departure then one less platform is needed to decrease the count, i.e., plat– and increment j
- Update the ans, i.e. ans = max(ans, plat).

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.print("Enter the length of the arrays: ");
       int n = scn.nextInt();
       int[] arr = new int[n];
       int[] dep = new int[n];
       for(int i = 0; i < n; i++){
           arr[i] = scn.nextInt();
       }
       for(int i = 0; i < n; i++){
           dep[i] = scn.nextInt();
       }
       Arrays.sort(arr);
       Arrays.sort(dep);
       int plat_needed = 1, result = 1;
       int i = 1, j = 0;
       while (i < n && j < n) {
           if (arr[i] <= dep[j]) {
               plat_needed++;//cannot be overlapped so need new platform
               i++;
           }else if (arr[i] > dep[j]) {
               plat_needed--;
               j++;
           }
           if (plat_needed > result)
               result = plat_needed;
       }
            System.out.print(result);
   }
}
```

```
6
0900 0940 0950 1100 1500 1800
0910 1200 1120 1130 1900 2000
3
Process finished with exit code 0
```