Greedy Questions

Question 1 :

Maximum Balanced String Partitions

We have balanced string str of size N with an equal number of L and R, the task is to find a maximum number X, such that a given string can be partitioned into X balanced substring. A string is called to be balanced if the number of 'L's in the string equals the number of 'R's.

Input : "LRRRRLLRLLRL"
Output : 3

Question 2 :

Kth largest odd number in a given range

We have two variables L and R, indicating a range of integers from L to R inclusive, and a number K, the task is to find Kth largest odd number. If K > number of odd numbers in the range L to R then return 0.

Sample Input 1 : L = -3, R = 3, K = 1
Sample Output 1 : 3

Question 3 :

Lexicographically smallest string of length N and sum K

We have two integers N and K. The task is to print the lexicographically smallest string of length N consisting of lower-case English alphabets such that the sum of the characters of the string equals to K where 'a' = 1, 'b' = 2, 'c' = 3, ... and 'z' = 26.

Sample Input 1 : N = 5, K = 42
Sample Output 1 : aamz

Sample Input 2 : N = 3, K = 25
Sample Output 2 : aaw

Question 4 :

Best Time to Buy and Sell Stock

Given an array prices[] of length N, representing the prices of the stocks on different days, the task is to find the maximum profit possible for buying and selling the stocks on different days using transactions where at most one transaction is allowed.
Note: Stock must be bought before being sold.

Sample Input 1 : `prices[] = {7, 6, 4, 3, 1}`
Sample Output 1 : 0

Sample Input 2 : `prices[] = {7, 1, 5, 3, 6, 4]`
Sample Output 2 : 5

Question 5 :

Split the given array into K sub-arrays

We have an Array[] of N elements and a number K. (1 <= K <= N) . Split the given array into K subarrays (they must cover all the elements). The maximum subarray sum achievable out of K subarrays formed must be the minimum possible. Find that possible subarray sum.

Sample Input 1 :Array[] = {1, 1, 2} K = 2
Sample Output 1 : 2

Sample Input 2 : `Array[] = {1, 2, 3, 4}, K = 3`
Sample Output 2 : 4

Comparators in Java

A **Comparator** in Java is an interface that is used to order the objects of user-defined classes. A comparator object is capable of comparing two objects of the same class.

Let's take an example of a function that compares obj1 with obj2 :
public int compare(Object obj1, Object obj2):

There are 2 ways of implementing this sort method -

a. We write a custom function on our own, where we define the sorting logic from scratch.
b. Using the Comparator interface. (Better!)

How to use sort?

Comparator interface is used to order/sort the objects of a user-defined class.

This interface is present in the java.util package and contains 2 methods compare(Object obj1, Object obj2) and equals(Object element).

Using a comparator, we can sort the elements based on data members/properties. For instance, it may be on the basis of name, age, height etc.

Method of Collections class for sorting List elements is used to sort the elements of List by the given comparator.
public void sort(List list, ComparatorClass c)

Internal working of the sort( ) method of Collections class

sort( ) method calls the Compare method of the classes it is sorting.
To compare 2 objects, it asks "Which is greater?"
Compare method returns one of the 3 values : -1, 0, or 1.

-1 -> obj1 is less than obj2
0 -> obj1 is equal to obj2
1 -> obj1 is greater than obj2
It uses this result to then determine if they(obj1 & obj2) should be swapped for their sort.

To define a **Customized Sorting Order**

```
import java.util.ArrayList;
import java.util.Collections;

public class Solution {
   public static void main (String[] args){
       ArrayList<Person> list = new ArrayList<Person>();
       list.add (new Person("Aman", 34));
       list.add (new Person("Akbar", 42));
       list.add (new Person("Anthony", 28));

       Collections.sort (list);
       System.out.println (list);
   }
}

class Person implements Comparable<Person>{
   String name;
   int age;

   Person(String name, int age){
       this.name = name;
       this.age = age;
   }

   /**
    * This function compares 2 objects
    * A person object is passed as an argument
    * and returns negative integer, zero, or a positive integer
    * as this object is less than, equal to, or greater than the specified object.
    */
   @Override
```

```java
   public int compareTo(Person person){
       if(this.age == person.age)
           return 0;
       else
           return (this.age < person.age) ? -1 : 1;
   }

   @Override
   public String toString(){
       return this.name + ":" + this.age;
   }
}
```

Lambda Expressions in Java

A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

Format:

Comparator<ClassName> comparator = Comparator.comparing(o -> o.property);

Example:

Comparator<Student> comparator = Comparator.comparing(o -> o.age);
Collections.sort(students, comparator);