Assignment Questions 10 Recursion

Question 1.
Given an integer ‘n‘, return *‘true‘ if it is a power of three. Otherwise, return ‘false‘*.
An integer ‘n‘ is a power of three, if there exists an integer ‘x‘ such that ‘n == 3x‘.

Example 1:
Input: n = 27
Output: true
Explanation: 27 = 33

Example 2:
Input: n = 0
Output: false
Explanation: There is no x where 3x = 0.

Example 3:
Input: n = -1
Output: false
Explanation: There is no x where 3x = (-1).

code :-

```
class Solution {
    public boolean isPowerOfThree(int n) {
        if(n<=0)
        return false;
        if(n==1)
        return true;
        if(n%3==0)
        return isPowerOfThree(n/3);
        else
        return false;
    }
}
```

Question 2
You have a list arr of all integers in the range [1, n] sorted in a strictly increasing order. Apply the following algorithm on arr:

- Starting from left to right, remove the first number and every other number afterward until you reach the end of the list.
- Repeat the previous step again, but this time from right to left, remove the rightmost number and every o ther number from the remaining numbers.
- Keep repeating the steps again, alternating left to right and right to left, until a single number remains.
Given the integer ‘n‘, return *the last number that remains in* ‘arr‘.

Example 1:
Input: n = 9
Output: 6
Explanation:
arr = [1, 2,3, 4,5, 6,7, 8,9]
arr = [2,4, 6,8]
arr = [2, 6]
arr = [6]

Example 2:
Input: n = 1
Output: 1

code :-

```
class Solution {
    public int lastRemaining(int n) {
        return doTheThing(n, 1, 2);
    }

    private int doTheThing(int size, int start, int direction) {
        if (size == 1) return start;           // one number left; must be the answer!

        int jumps = size / 2 + (size % 2) - 1;  // quick and dirty ceil(size/2 - 1);
                                                // this is the number of legal hops from
                                                // one end of the remaining array to the other.

        int end = start + (direction * jumps);  // figure out the last number eliminated

        // If the array is odd sized (size % 2 == 1, end is the last element of the array, so the
        // first number that hasn't been eliminated is exactly between end and (end - direction).
        // If the array is even size (size % 2 == 0), then there's exactly 1 element in the direction
        // we're moving after end. The distance is always going to be half the hop size.
        int nextStart = end - ((2 * (size % 2) - 1) * direction/2);

        // We've effectively culled half the array and figured out the number on one end of the array
        // which has not been eliminated. Repeat until there's one left.
        return doTheThing(size/2, nextStart, direction * -2);
    }
}
```

Question 3.
Given a set represented as a string, write a recursive code to print all subsets of it. The subsets can be pri nted in any order.
Example 1:
Input : set = “abc”
Output : { “”, “a”, “b”, “c”, “ab”, “ac”, “bc”, “abc”}

Example 2:
Input : set = “abcd”
Output : { “”, “a” ,”ab” ,”abc” ,”abcd”, “abd” ,”ac” ,”acd”, “ad” ,”b”, “bc” ,”bcd” ,”bd” ,”c” ,”cd” ,”d” }

code:-

```
import java.util.*;
import java.io.*;

public class Main {
    //Function of sorting a string
  public static String sortString(String inputString){
        // convert input string to char array
        char tempArray[] = inputString.toCharArray();
        // sort tempArray
```

```
        Arrays.sort(tempArray);
        // return new sorted string
        return new String(tempArray);
    }
  static void powerSet(String s,String cur,int index){
    int n = s.length();
    if(index == n)
      return ;
    System.out.println(cur);

    for(int i = index+1; i<n; i++){
      cur += s.charAt(i);
      /* After all subset beginning with "cur" are printed,
      we will remove the last character to consider different
      prefix of subsets */
      powerSet(s,cur,i);
      cur = cur.substring(0, cur.length() - 1);
    }
}
  public static void main(String args[]) throws IOException {
    Scanner sc = new Scanner(System.in);
    String str = sc.next();
    str = sortString(str);
    String cur = "";
    int index = -1;
    powerSet(str,cur,index);
  }
}
```

Question 4:
Given a string calculate length of the string using recursion.
Examples:
Input : str = "abcd"
Output :4

Input : str = "GEEKSFORGEEKS"
Output :13

code:-

```
public class StringLength{
 private static int recLen(String str)
 {
  if (str.equals(""))
   return 0;
  else
   return recLen(str.substring(1)) + 1;
 }
 public static void main(String[] args)
 {
  String str ="abcd";
  System.out.println(recLen(str));
 }
}
```

Question 5.
We are given a string S, we need to find count of all contiguous substrings starting and ending with same character.
Examples :
Input : S = "abcab"
Output : 7
There are 15 substrings of "abcab"
a, ab, abc, abca, abcab, b, bc, bca
bcab, c, ca, cab, a, ab, b
Out of the above substrings, there
are 7 substrings : a, abca, b, bcab,
c, a and b.

Input : S = "aba"
Output : 4
The substrings are a, b, a and aba

code:-

```
public class SubString1 {
 static final int MAX_CHAR = 26;
 static int countSubstringWithEqualEnds(String s)
 {
  int result = 0;
  int n = s.length();
  int[] count = new int[MAX_CHAR];
  for (int i = 0; i < n; i++)
   count[s.charAt(i)-'a']++;
  for (int i = 0; i < MAX_CHAR; i++)
   result += (count[i] * (count[i] + 1) / 2);
  return result;
 }
  public static void main(String args[])
 {
  String s = "abcab";
  System.out.println(countSubstringWithEqualEnds(s));
 }
}
```

Question 6:
The [tower of Hanoi](https://en.wikipedia.org/wiki/Tower_of_Hanoi) is a famous puzzle where we have thr ee rods and **N** disks. The objective of the puzzle is to move the entire stack to another rod. You are gi ven the number of discs **N**. Initially, these discs are in the rod 1. You need to print all the steps of discs movement so that all the discs reach the 3rd rod. Also, you need to find the total moves.**Note:** The dis cs are arranged such that the **top disc is numbered 1** and the **bottom-most disc is numbered N**. Als o, all the discs have **different sizes** and a bigger disc **cannot** be put on the top of a smaller disc. Re fer the provided link to get a better clarity about the puzzle.
Example 1:
Input:
N = 2
Output:
move disk 1 from rod 1 to rod 2
move disk 2 from rod 1 to rod 3
move disk 1 from rod 2 to rod 3
3

Explanation:For N=2 , steps will be
as follows in the example and total
3 steps will be taken.

Example 2:
Input:
N = 3
Output:
move disk 1 from rod 1 to rod 3
move disk 2 from rod 1 to rod 2
move disk 1 from rod 3 to rod 2
move disk 3 from rod 1 to rod 3
move disk 1 from rod 2 to rod 1
move disk 2 from rod 2 to rod 3
move disk 1 from rod 1 to rod 3
7
Explanation:For N=3 , steps will be
as follows in the example and total
7 steps will be taken.

code:-

```
class Main
{
 public static void move(int disks, int source, int auxiliary, int target)
 {
  if (disks > 0)
  {
   move(disks - 1, source, target, auxiliary);
   System.out.println("Move disk " + disks + " from " + source + "+target);

   move(disks - 1, auxiliary, source, target);
  }
 }
 public static void main(String[] args)
 {
  int n = 3;
  move(n, 1, 2, 3);
 }
}
```

Question 7.
Given a string **str**, the task is to print all the permutations of **str**. A **permutation** is an arrangeme
nt of all or part of a set of objects, with regard to the order of the arrangement. For instance, the words ‘ba
t’ and ‘tab’ represents two distinct permutation (or arrangements) of a similar three letter word.
Examples:

Input: str = “cd”
Output:cd dc
Input: str = “abb”
Output: abb abb bab bba bab bba

code:-

```
class Main {
    static void printPermutn(String str, String ans) {
        if (str.length() == 0) {
```

```
            System.out.print(ans + " ");
            return;
        }
        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            String r = str.substring(0, i) + str.substring(i + 1);
            printPermutn(r, ans + ch);
        }
    }
    public static void main(String[] args) {
        String s = "abb";
        printPermutn(s, "");
    }
}
```

Question 8
Given a string, count total number of consonants in it. A consonant is an English alphabet character that i s not vowel (a, e, i, o and u). Examples of constants are b, c, d, f, and g.
Examples :
Input : abc de
Output : 3
There are three consonants b, c and d.

Input : geeksforgeeks portal
Output : 12

code:-

```
import java.util.*;
import java.lang.*;

class VowelConsonent
{
static boolean isConsonant(char ch)
{
 ch = Character.toUpperCase(ch);
 return (ch == 'A' || ch == 'E' ||
   ch == 'I' || ch == 'O' ||
   ch == 'U')== false && ch >= 65 && ch <= 90;
}
static int totalConsonants(String str, int n)
{
 if (n == 1)
 {
  if(isConsonant(str.charAt(0)))
   return 1;
  else
   return 0;
 }

 if(isConsonant(str.charAt(n - 1)))
  return totalConsonants(str, n - 1) + 1;
 else
  return totalConsonants(str, n - 1);
}
public static void main(String args[])
```

```
{
 String str = "abc de";
 System.out.println(totalConsonants(str, str.length()));
}
}
```