Recursion - II

```
Remove X
Send Feedback
Given a string, compute recursively a new string where all 'x' chars have
been removed.
Input format :

String S

Output format :

Modified String

Constraints :

1 <= |S| <= 10^3
where |S| represents the length of string S.

Sample Input 1 :

xaxb

Sample Output 1:

ab

Sample Input 2 :

abc

Sample Output 2:

abc

Sample Input 3 :

xx

Sample Output 3:

public class solution {

   // Return the changed string
   public static String removeX(String input){
       // Write your code here
       if(input.length()==0)
           return input;
       String smalloutput = removeX(input.substring(1));
       if(input.charAt(0)=='x')
           return smalloutput;
       else
           return input.charAt(0)+smalloutput;
```

```
    }
}
```

Replace Characters Recursively
Send Feedback
Given an input string *S* and two characters c1 and c2, you need to replace every occurrence of character c1 with character c2 in the given string.
Do this recursively.
Input Format :

Line 1 : *Input String S*
Line 2 : *Character* c1 and c2 (separated by space)

Output Format :

Updated string

Constraints :
1 <= *Length* of *String S* <= 10^6
Sample Input :

```
abacd
a x
```

Sample Output :

```
xbxcd
```

```
public class Solution {

   public static String replaceCharacter(String input, char c1, char c2) {
       if(input.length()==0)
           return input;

       if(input.charAt(0)==c1)
           return c2+replaceCharacter(input.substring(1), c1, c2);
       else
           return input.charAt(0)+replaceCharacter(input.substring(1), c1,
c2);
   }
}
```

Remove Duplicates Recursively
Send Feedback
Given a string S, remove consecutive duplicates from it recursively.
Input Format :

String S

Output Format :

Output string

Constraints :

```
1 <= |S| <= 10^3
where |S| represents the length of string

Sample Input 1 :

aabccba

Sample Output 1 :

abcba

Sample Input 2 :

xxxyyyzwwzzz

Sample Output 2 :

xyzwz

public class Solution {

   public static String removeConsecutiveDuplicates(String s) {
       // Write your code here
       if(s.length()<=1)
           return s;

       if(s.charAt(0)==s.charAt(1))
           return removeConsecutiveDuplicates(s.substring(1));
       else
           return s.charAt(0)+removeConsecutiveDuplicates(s.substring(1));

       // String smalloutput =
removeConsecutiveDuplicates(s.substring(1));
       // if(s.charAt(0)==smalloutput.charAt(0))
       //     return smalloutput;
   }
}
```

```
Merge Sort - Problem Statement
Send Feedback
Sort an array A using Merge Sort.
Change in the input array itself. So no need to return or print anything.
Input format :

Line 1 : Integer n i.e. Array size
Line 2 : Array elements (separated by space)

Output format :

Array elements in increasing order (separated by space)

Constraints :
```

```
1 <= n <= 10^3

Sample Input 1 :

6
2 6 8 5 4 3

Sample Output 1 :

2 3 4 5 6 8

Sample Input 2 :

5
2 1 5 2 3

Sample Output 2 :

1 2 2 3 5


public class solution {

   public static void mergeSort(int[] input){
       // Write your code here
       if(input.length==1)
           return;

       int mid = input.length%2==0? input.length/2: input.length/2+1;

       int lefthalf[] = new int[mid];
       int righthalf[] = new int[input.length-mid];

       for(int i=0; i<mid; i++){
           lefthalf[i] = input[i];
       }
       for(int temp=0, i=mid; i<input.length; temp++,i++){
           righthalf[temp] = input[i];
       }

       mergeSort(lefthalf);
       mergeSort(righthalf);
       // int res[] = new int[lefthalf.length*2];
       int i=0, j=0, k = 0;

       while(i<lefthalf.length && j<righthalf.length){
           if(lefthalf[i]<righthalf[j])
               input[k++] = lefthalf[i++];
           else
               input[k++] = righthalf[j++];

       }
       while(i<lefthalf.length)
           input[k++] = lefthalf[i++];
```

```
        while(j<righthalf.length)
            input[k++] = righthalf[j++];

    }
}
```

```
Quick Sort - Problem Statement
Send Feedback
Sort an array A using Quick Sort.
Change in the input array itself. So no need to return or print anything.


Input format :

Line 1 : Integer n i.e. Array size
Line 2 : Array elements (separated by space)

Output format :

Array elements in increasing order (separated by space)

Constraints :

1 <= n <= 10^3

Sample Input 1 :

6
2 6 8 5 4 3

Sample Output 1 :

2 3 4 5 6 8

Sample Input 2 :

5
1 5 2 7 3

Sample Output 2 :

1 2 3 5 7

public class Solution {

    public static int partition(int[] input, int si, int ei) {

        int pivotelement = input[si];
        int count = 0;
        for(int i=si; i<=ei; i++){
            if(input[i]<pivotelement)
                count++;
        }
```

```
        int temp = input[si+count];
        input[si+count] = pivotelement;
        input[si] = temp;

        int i=si, j=ei;
        while(i<j){
            if(input[i]<pivotelement)
                i++;
            else if(input[j]>=pivotelement)
                j--;
            else{
                temp = input[j];
                input[j] = input[i];
                input[i] = temp;
                i++; j--;
            }
        }
        return si+count;
    }

    public static void sort(int[] input, int si, int ei){
        if(si>=ei){
            return;
        }

        int pivotelement = partition(input, si, ei);
        sort(input, si, pivotelement-1);
        sort(input, pivotelement+1, ei);
    }

    public static void quickSort(int[] input){
        sort(input, 0, input.length-1);
    }

}
```

Tower Of Hanoi - Problem Statement
Send Feedback
Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move all disks from source rod to destination rod using third rod (say auxiliary). The rules are :

1) Only one disk can be moved at a time.
2) A disk can be moved only if it is on the top of a rod.
3) No disk can be placed on the top of a smaller disk.

Print the steps required to move n disks from source rod to destination rod.
Source Rod is named as 'a', auxiliary rod as 'b' and destination rod as 'c'.
Input Format :

Integer n

Output Format :

Steps in different lines (in one line print source and destination rod name separated by space)

Constraints :

0 <= n <= 20

Sample Input 1 :

2

Sample Output 1 :

a b
a c
b c

Sample Input 2 :

3

Sample Output 2 :

a c
a b
c b
a c
b a
b c
a c

```
public class solution {
   public static void towerOfHanoi(int n,char s,char h,char d){
       if (n == 0)
       {
           return;
       }
       if(n==1){
           System.out.println(s+" "+d);
           return;
       }
       towerOfHanoi(n-1,s,d,h);
       System.out.println(s+" "+d);
       towerOfHanoi(n-1,h,s,d);
   }
}
```