6. ZigZag Conversion

The string `"PAYPALISHIRING"` is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

```
P   A   H   N
A P L S I I G
Y   I   R
```

And then read line by line: `"PAHNAPLSIIGYIR"`

Write the code that will take a string and make this conversion given a number of rows:

```
string convert(string s, int numRows);
```

Example 1:

```
Input: s = "PAYPALISHIRING", numRows = 3
Output: "PAHNAPLSIIGYIR"
```

Example 2:

```
Input: s = "PAYPALISHIRING", numRows = 4
Output: "PINALSIGYAHRPI"
Explanation:
P     I    N
A   L S  I G
Y A   H R
P     I
```

Example 3:

```
Input: s = "A", numRows = 1
Output: "A"
```

Constraints:

- `1 <= s.length <= 1000`
- `s` consists of English letters (lower-case and upper-case), `','` and `'.'`.
- `1 <= numRows <= 1000`

https://leetcode.com/problems/zigzag-conversion/submissions/ (https://leetcode.com/problems/zigzag-conversion/submissions/) **My Solution**

```
class Solution {
    public String convert(String s, int numRows) {
        if(s.length()==0)
            return "";
        if(numRows==1)
            return s;
        StringBuffer ans=new StringBuffer();
        ArrayList<ArrayList<Character>> l=new ArrayList<ArrayList<Character>>();
        int divi=2*numRows-2;
        for(int j=0;j<numRows;j++)
        {
            l.add(new ArrayList<Character>());
        }
        int p=0;
        for(int i=0;i<s.length()/divi;i++)
        {
            for(int j=0;j<numRows;j++)
            {
                l.get(j).add(s.charAt(p));
                p++;
            }
            for(int j=numRows-2;j>=1;j--)
            {
                l.get(j).add(s.charAt(p));
                p++;
            }
        }
        if(s.length()%divi>numRows)
        {
            for(int j=0;j<numRows;j++)
            {
                l.get(j).add(s.charAt(p));
                p++;
            }
            for(int j=numRows-2;j>2*numRows-2-s.length()%divi;j--)
            {
                l.get(j).add(s.charAt(p));
                p++;
            }

        }
        else
        {
        for(int i=0;i<s.length()%divi;i++)
        {
            l.get(i).add(s.charAt(p));
            p++;
        }
```

```
        }
        for(ArrayList<Character> t:l)
        {
            for(Character temp:t)
            {
                ans.append(temp);
            }
        }
        return  ans.toString();
    }
}
```

Simplified My solution

```
class Solution {
    public String convert(String s, int numRows) {

        if (numRows == 1) return s;

        StringBuilder ret = new StringBuilder();
        int n = s.length();
        int cycleLen = 2 * numRows - 2;

        for (int i = 0; i < numRows; i++) {
            for (int j = 0; j + i < n; j += cycleLen) {
                ret.append(s.charAt(j + i));
                if (i != 0 && i != numRows - 1 && j + cycleLen - i < n)
                    ret.append(s.charAt(j + cycleLen - i));
            }
        }
        return ret.toString();
    }
}
```

LeetCode Solution

```
class Solution {
    public String convert(String s, int numRows) {

        if (numRows == 1) return s;

        List<StringBuilder> rows = new ArrayList<>();
        for (int i = 0; i < Math.min(numRows, s.length()); i++)
            rows.add(new StringBuilder());

        int curRow = 0;
        boolean goingDown = false;

        for (char c : s.toCharArray()) {
            rows.get(curRow).append(c);
            if (curRow == 0 || curRow == numRows - 1) goingDown = !goingDown;
            curRow += goingDown ? 1 : -1;
        }

        StringBuilder ret = new StringBuilder();
        for (StringBuilder row : rows) ret.append(row);
        return ret.toString();
    }
}
```

---

8. String to Integer (atoi)

Implement `atoi` which converts a string to an integer.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str is not a valid integral number, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.

If no valid conversion could be performed, a zero value is returned.

Note:

- Only the space character `' '` is considered a whitespace character.
- Assume we are dealing with an environment that could only store integers within the 32-bit signed integer range: $[-2^{31}, 2^{31} - 1]$. If the numerical value is out of the range of representable values, INT_MAX ($2^{31} - 1$) or INT_MIN (-2^{31}) is returned.

Example 1:

```
Input: str = "42"
Output: 42
```

Example 2:

```
Input: str = "   -42"
Output: -42
Explanation: The first non-whitespace character is '-', which is the minus sign. Then t
```

Example 3:

```
Input: str = "4193 with words"
Output: 4193
Explanation: Conversion stops at digit '3' as the next character is not a numerical dig
```

Example 4:

```
Input: str = "words and 987"
Output: 0
Explanation: The first non-whitespace character is 'w', which is not a numerical digit
```

Example 5:

```
Input: str = "-91283472332"
Output: -2147483648
Explanation: The number "-91283472332" is out of the range of a 32-bit signed integer.
```

Constraints:

- `0 <= s.length <= 200`
- `s` consists of English letters (lower-case and upper-case), digits, `' '`, `'+'`, `'-'` and `'.'`.

---

https://leetcode.com/problems/string-to-integer-atoi/ (https://leetcode.com/problems/string-to-integer-atoi/) problem no. 8 **My solution**

```
class Solution {
    public int myAtoi(String str) {
        int n=str.length();
        int i=0,ans=0,flags=0,flagsp=0;
        char sign='+';
        ArrayList<Integer> s=new ArrayList<Integer>();
        while(i!=n)
        {
            char current=str.charAt(i);
            if(current==' ')
            {
                if(flagsp==1)
                    return value(s,sign);
                i++;
                continue;
            }
            else if(current=='+' || current=='-')
            {
                if(flags==0)
                    sign=current;
                else if(flags==1)
                    return value(s,sign);
                flags=1;
                flagsp=1;
            }
            else if(current>=48 && current<=57)
            {
                s.add(Integer.valueOf(current-48));
                flagsp=1;
                flags=1;
            }
            else
            {
                return value(s,sign);
            }
            i++;
        }
        return value(s,sign);
    }
    public int value(ArrayList<Integer> num,char sign)
    {
        long ans=0;
        long x=1;
        while(num.size()>0 && num.get(0)==0)  //remove excees 0 at start
        {
            num.remove(0);
        }
        if(num.size()>10)    // if value is excedding long for test case 100000000000
```

```
00000000000000000000000000000000034536536
        {
            if(sign=='+')
                return Integer.MAX_VALUE;
            else
                return Integer.MIN_VALUE;
        }

        for(int i=num.size()-1;i>=0;i--)
        {
            // System.out.println(num.get(i));
            long val;
            if(sign=='+')
                val=ans+num.get(i)*x;
            else
                 val=ans-num.get(i)*x;
            if(val>Integer.MAX_VALUE)
                return Integer.MAX_VALUE;
            if(val<Integer.MIN_VALUE)
                return Integer.MIN_VALUE;
            ans=val;
            x=x*10;
        }
        return (int)ans;
    }
}
```

clean code

```
class Solution {
    public int myAtoi(String str) {
        int i = 0;
        int sign = 1;
        int result = 0;
        if (str.length() == 0) return 0;

        //Discard whitespaces in the beginning
        while (i < str.length() && str.charAt(i) == ' ')
            i++;

        // Check if optional sign if it exists
        if (i < str.length() && (str.charAt(i) == '+' || str.charAt(i) == '-'))
            sign = (str.charAt(i++) == '-') ? -1 : 1;

        // Build the result and check for overflow/underflow condition
        while (i < str.length() && str.charAt(i) >= '0' && str.charAt(i) <= '9') {
            if (result > Integer.MAX_VALUE / 10 ||
                    (result == Integer.MAX_VALUE / 10 && str.charAt(i) - '0' > Intege
r.MAX_VALUE % 10)) {
                return (sign == 1) ? Integer.MAX_VALUE : Integer.MIN_VALUE;
            }
            result = result * 10 + (str.charAt(i++) - '0');
        }
        return result * sign;

    }
}
```

14. Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string `""`.

Example 1:

```
Input: strs = ["flower","flow","flight"]
Output: "fl"
```

Example 2:

```
Input: strs = ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.
```

Constraints:

- `0 <= strs.length <= 200`
- `0 <= strs[i].length <= 200`
- `strs[i]` consists of only lower-case English letters.

---

https://leetcode.com/problems/longest-common-prefix/ (https://leetcode.com/problems/longest-common-prefix/) /* Explanation:

1) find minimum length word(temp) from given list 2) compare each letter of temp word with respective letters of all other words in list append letter in ans if letter of temp(small length word) found in each word from list / *My solution**

```
class Solution {
    public String longestCommonPrefix(String[] strs) {
        if(strs.length==0)
          return "";
        StringBuffer s=new StringBuffer();
        int min=Integer.MAX_VALUE;
        String temp="";
        for(int i=0;i<strs.length;i++)
        {
            if(min>strs[i].length())
            {
                min=strs[i].length();
                temp=strs[i];
            }
        }
        for(int i=0;i<min;i++)
        {
            int j;
            for(j=0;j<strs.length;j++)
            {
                if(strs[j].charAt(i)!=temp.charAt(i))
                {
                    break;
                }
            }
            if(j==strs.length)
                s.append(temp.charAt(i));
            else
                break;
        }
        return s.toString();
    }
}
```

20. Valid Parentheses

Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Example 1:

```
Input: s = "()"
Output: true
```

Example 2:

```
Input: s = "()[]{}"
Output: true
```

Example 3:

```
Input: s = "(]"
Output: false
```

Example 4:

```
Input: s = "([)]"
Output: false
```

Example 5:

```
Input: s = "{[]}"
Output: true
```

Constraints:

- `1 <= s.length <= 10^4`
- `s` consists of parentheses only `'()[]{}'`.

---

https://leetcode.com/problems/valid-parentheses/ (https://leetcode.com/problems/valid-parentheses/) problem no 20 **My solution**

```
class Solution {
    public boolean isValid(String s) {
        Stack <Character> st =new Stack<Character>();
        for(int i=0;i<s.length();i++)
        {
            if(s.charAt(i)=='(' || s.charAt(i)=='{' || s.charAt(i)=='[')
                st.push(s.charAt(i));
            else
            {
                if(st.empty())
                    return false;
                char temp=st.pop();
                if(s.charAt(i)==')' && temp!='(')
                {
                    return false;
                }
                else if(s.charAt(i)=='}' && temp!='{')
                {
                    return false;
                }
                else if(s.charAt(i)==']' && temp!='[')
                {
                    return false;
                }
            }
        }
        if(!st.empty())
            return false;
        return true;
    }
}
```

Alternative solution

```
public boolean isValid(String s) {

     if(s.length() % 2 == 1)
        return false;

     Stack<Character> stack = new Stack<Character>();
     for(int i = 0; i < s.length(); i++)
     {
        if(s.charAt(i) == '(')
        {
            stack.push(')');
        }
        else if(s.charAt(i) == '{')
        {
            stack.push('}');
        }
        else if(s.charAt(i) == '[')
        {
            stack.push(']');
        }
        else if(stack.isEmpty() || stack.pop() != s.charAt(i))
            return false;
     }
     return stack.isEmpty();
}
```

28. Implement strStr()

Implement strStr() (http://www.cplusplus.com/reference/cstring/strstr/).

Return the index of the first occurrence of needle in haystack, or `-1` if `needle` is not part of `haystack`.

Clarification:

What should we return when `needle` is an empty string? This is a great question to ask during an interview.

For the purpose of this problem, we will return 0 when `needle` is an empty string. This is consistent to C's strstr() (http://www.cplusplus.com/reference/cstring/strstr/) and Java's indexOf() (https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#indexOf(java.lang.String)).

Example 1:

```
Input: haystack = "hello", needle = "ll"
Output: 2
```

Example 2:

```
Input: haystack = "aaaaa", needle = "bba"
Output: -1
```

Example 3:

```
Input: haystack = "", needle = ""
Output: 0
```

Constraints:

- `0 <= haystack.length, needle.length <= 5 * 10^4`
- `haystack` and `needle` consist of only lower-case English characters.

---

https://leetcode.com/problems/implement-strstr/ (https://leetcode.com/problems/implement-strstr/)
problem no : 28 **Brute Force**

```
class Solution {
    public int strStr(String haystack, String needle) {
        int n=haystack.length();
        int m=needle.length();
        if(m>n)
            return -1;
        if(m==0)
            return 0;
        for(int i=0;i<n;i++)
        {
            if(haystack.charAt(i)==needle.charAt(0) && i<=n-m)
            {
                int j;
                for(j=1;j<m;j++)
                {
                    if(haystack.charAt(i+j)!=needle.charAt(j))
                        break;
                }
                if(j==m)
                    return i;
            }
        }
        return -1;
    }
}
```

38. Count and Say

The count-and-say sequence is the sequence of integers with the first five terms as following:

```
1.     1
2.     11
3.     21
4.     1211
5.     111221
```

`1` is read off as `"one 1"` or `11`.
`11` is read off as `"two 1s"` or `21`.
`21` is read off as `"one 2`, then `one 1"` or `1211`.

Given an integer *n* where 1 ≤ *n* ≤ 30, generate the n^{th} term of the count-and-say sequence. You can do so recursively, in other words from the previous member read off the digits, counting the number of digits in groups of the same digit.

Note: Each term of the sequence of integers will be represented as a string.

Example 1:

```
Input: 1
Output: "1"
Explanation: This is the base case.
```

Example 2:

```
Input: 4
Output: "1211"
Explanation: For n = 3 the term was "21" in which we have two groups "2" and "1", "2" c
```

https://leetcode.com/problems/count-and-say/ (https://leetcode.com/problems/count-and-say/) problem no 38 /* Explanation: Use recursion as answer is dependent on previous answer function pro is returning required answer for n-1 to produce answer for n / *My solution**

```
class Solution {
    public String countAndSay(int n) {
        if(n==1)
            return "1";
        return pro(countAndSay(n-1).toCharArray());
    }
    public String pro(char []s)
    {
        int n=s.length;
        StringBuffer ans=new StringBuffer();
        int count=1;
        for(int i=1;i<n;i++)
        {
            if(s[i]!=s[i-1])
            {
                ans.append(count);
                ans.append(s[i-1]);
                count=1;
            }
            else
            {
                count++;
            }
        }
        ans.append(count);
        ans.append(s[n-1]);
        return ans.toString();
    }
}
```

---

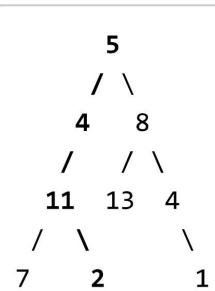
112. Path Sum

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

Note: A leaf is a node with no children.

Example:

Given the below binary tree and `sum = 22`,



return true, as there exist a root-to-leaf path `5->4->11->2` which sum is 22.

---

125. Valid Palindrome

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

Note: For the purpose of this problem, we define empty string as valid palindrome.

Example 1:

```
Input: "A man, a plan, a canal: Panama"
Output: true
```

Example 2:

```
Input: "race a car"
Output: false
```

Constraints:

- `s` consists only of printable ASCII characters.

---

https://leetcode.com/problems/valid-palindrome/ (https://leetcode.com/problems/valid-palindrome/) problem no 125 2 user defined function

1) isnumalpha - check whether alphabet or numerical 2) casesense- it will return small alphabet chacater to Capital alphabet character i.e. a ->A , b->B .... will not affect Capital letters and number

```
class Solution {
    public boolean isPalindrome(String s) {
        int n=s.length();
        int i=0,j=n-1;
        while(i<=j)
        {
            char ichar=s.charAt(i);
            char jchar=s.charAt(j);
            if(!isnumalpha(ichar))  // check whether char at i is alphanumeric or not
            {
                i++;
                continue;
            }
            if(!isnumalpha(jchar))   // check whether char at j is alphanumeric or no
t
            {
                j--;
                continue;
            }
            if(casesense(ichar)==casesense(jchar)) // if both char at respective poin
ter equal then increment both pointer
            {
                i++;
                j--;
            }
            else
                return false;
        }
        return true;
    }
    public boolean isnumalpha(char c)
    {
        if((c>=65 && c<=90) || (c>=97 && c<=122))
            return true;
        else if(c>='0' && c<='9')
            return true;
        return false;
    }
    public char casesense(char c)
    {
        if(c>=97 && c<=122)   // if alphabet is small letter -32 is done to convert i
t into Capital letter (See ASCII Value)
            return (char)(c-32);
        return c;
    }
}
```

128. Longest Consecutive Sequence

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

Your algorithm should run in O(*n*) complexity.

Example:

```
Input: [100, 4, 200, 1, 3, 2]
Output: 4
Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its l
```

151. Reverse Words in a String

Given an input string `s`, reverse the order of the **words**.

A **word** is defined as a sequence of non-space characters. The **words** in `s` will be separated by at least one space.

Return *a string of the words in reverse order concatenated by a single space.*

Note that `s` may contain leading or trailing spaces or multiple spaces between two words. The returned string should only have a single space separating the words. Do not include any extra spaces.

Example 1:

```
Input: s = "the sky is blue"
Output: "blue is sky the"
```

Example 2:

```
Input: s = "  hello world  "
Output: "world hello"
Explanation: Your reversed string should not contain leading or trailing spaces.
```

Example 3:

```
Input: s = "a good   example"
Output: "example good a"
Explanation: You need to reduce multiple spaces between two words to a single space in
```

Example 4:

```
Input: s = "  Bob    Loves  Alice   "
Output: "Alice Loves Bob"
```

Example 5:

```
Input: s = "Alice does not even like bob"
Output: "bob like even not does Alice"
```

Constraints:

- `1 <= s.length <= 10^4`
- `s` contains English letters (upper-case and lower-case), digits, and spaces `' '`.
- There is **at least one** word in `s`.

Follow up:

- Could you solve it **in-place** with `O(1)` extra space?

---

https://leetcode.com/problems/reverse-words-in-a-string/ (https://leetcode.com/problems/reverse-words-in-a-string/) **My solution**

```
class Solution {
    public String reverseWords(String s) {
        //final answer will store in ans
        StringBuilder ans=new StringBuilder();
        // it will store each word as it is till we find space char
        StringBuilder temp=new StringBuilder();
       // flag is to check whether no more than 2 space should between two words
        int flag=0;
        for(int i=0;i<s.length();i++)
        {
            if(s.charAt(i)!=' ')
            {
                temp.append(s.charAt(i));
                flag=0; // as we need one space after each word
            }
            else
            {
                if(flag==0) //if flag is 0 then space is found for first time
                 //so insert previous one word at starting
                {
                    ans.insert(0,temp);
                    temp=new StringBuilder();
                    if(i!=0 && i!=s.length()-1)// to check space is not at end or sta
rt
                    {
                        ans.insert(0,' '); // one space insertion
                    }
                }
                flag=1; // it indicate one space already added
            }
        }
        ans.insert(0,temp); // if we not found any space at end
        return ans.toString().trim();
// trim for case when thera are more than one space at end
    }
}
```

Without using trim function

```
class Solution {
    public String reverseWords(String s) {
        StringBuilder ans=new StringBuilder();
        StringBuilder temp=new StringBuilder();
        int flag=0;
        for(int i=0;i<s.length();i++)
        {
            if(s.charAt(i)!=' ')
            {
                temp.append(s.charAt(i));
                flag=0;
            }
            else
            {
                if(flag==0)
                {
                    if(ans.length()!=0)
                       ans.insert(0,' ');
                    ans.insert(0,temp);
                    temp=new StringBuilder();

                }
                flag=1;
            }
        }
        if(temp.length()!=0 && ans.length()!=0)
            ans.insert(0,' ');
        ans.insert(0,temp);
        return ans.toString();
    }
}
```

409. Longest Palindrome

Given a string `s` which consists of lowercase or uppercase letters, return *the length of the* ***longest palindrome*** that can be built with those letters.

Letters are **case sensitive**, for example, `"Aa"` is not considered a palindrome here.

Example 1:

```
Input: s = "abccccdd"
Output: 7
Explanation:
One longest palindrome that can be built is "dccaccd", whose length is 7.
```

Example 2:

```
Input: s = "a"
Output: 1
```

Example 3:

```
Input: s = "bb"
Output: 2
```

Constraints:

- `1 <= s.length <= 2000`
- `s` consits of lower-case **and/or** upper-case English letters only.

---

https://leetcode.com/problems/longest-palindrome/ (https://leetcode.com/problems/longest-palindrome/) problem no 409 *My solution *

```
class Solution {
    public int longestPalindrome(String s) {
        int sum=0;
        int freq[]=new int[59];
        for(int i=0;i<s.length();i++)
        {
            freq[s.charAt(i)-'A']++;
        }
        for(int i=0;i<59;i++)
        {
            if(freq[i]%2!=0)
                sum+=freq[i]-1;
            else
                sum+=freq[i];
        }
        if(sum!=s.length())
            sum++;
        return sum;
    }
}
```

Using Hashset only

```
class Solution {
    public int longestPalindrome(String s) {
        int sum=0;
        HashSet <Character> freq=new HashSet<Character>();
        for(int i=0;i<s.length();i++)
        {
            char current=s.charAt(i);
            if(freq.contains(current))
            {
                sum+=2;
                freq.remove(current);
            }
            else
               freq.add(current);
        }

        if(sum!=s.length())
            sum++;
        return sum;
    }
}
```

537. Complex Number Multiplication

Given two strings representing two complex numbers (https://en.wikipedia.org/wiki/Complex_number).

You need to return a string representing their multiplication. Note $i^2 = -1$ according to the definition.

Example 1:

```
Input: "1+1i", "1+1i"
Output: "0+2i"
Explanation: (1 + i) * (1 + i) = 1 + i^2 + 2 * i = 2i, and you need convert it to the fo
```

Example 2:

```
Input: "1+-1i", "1+-1i"
Output: "0+-2i"
Explanation: (1 - i) * (1 - i) = 1 + i^2 - 2 * i = -2i, and you need convert it to the f
```

Note:

1. The input strings will not have extra blank.
2. The input strings will be given in the form of **a+bi**, where the integer **a** and **b** will both belong to the range of [-100, 100]. And **the output should be also in this form**.

---

https://leetcode.com/problems/complex-number-multiplication/ (https://leetcode.com/problems/complex-number-multiplication/) problem no. 537

```
class Solution {
    public String complexNumberMultiply(String a, String b) {
        int first_real=0,second_real=0,first_complex=0,second_complex=0;
        int i=0;
        while(a.charAt(i)!='+') // for real number
        {
            i++;
        }
        first_real=Integer.parseInt(a.substring(0,i));
        int j=++i;    //Starting of imaginary part at index j
        while(a.charAt(i)!='i')  // for imaginary part
        {
            i++;
        }
        first_complex=Integer.parseInt(a.substring(j,i));


        i=0;
        while(b.charAt(i)!='+')
        {
            i++;
        }
        second_real=Integer.parseInt(b.substring(0,i));
        int k=++i;
        while(b.charAt(i)!='i')
        {
            i++;
        }
        second_complex=Integer.parseInt(b.substring(k,i));


        StringBuilder ans=new StringBuilder();
        int real_part=first_real*second_real-first_complex*second_complex;
        int imaginary_part=first_real*second_complex+first_complex*second_real;

        ans.append(real_part);
        ans.append('+');
        ans.append(imaginary_part);
        ans.append('i');

        return ans.toString();
    }
}
```

680. Valid Palindrome II

Given a non-empty string s , you may delete **at most** one character. Judge whether you can make it a palindrome.

Example 1:

```
Input: "aba"
Output: True
```

Example 2:

```
Input: "abca"
Output: True
Explanation: You could delete the character 'c'.
```

Note:

1. The string will only contain lowercase characters a-z. The maximum length of the string is 50000.

---

https://leetcode.com/problems/valid-palindrome-ii/ (https://leetcode.com/problems/valid-palindrome-ii/) problem no 680 1)Bruteforce - at every deletion check whether string is palindrome or not 2)Greedy Approach if we found mismatch char at i in given string the two possibilities of string is palindrome 1. string with char i to n-1-i-1 should be palindrome 2. string with char i+1 to n-1-i should be paindrome test case :"ebcbbececabbacecbbcbe"

```
class Solution {
    public boolean validPalindrome(String s) {
        int n=s.length();
        for(int i=0;i<n/2;i++)
        {
            if(s.charAt(i)!=s.charAt(n-1-i))
                return ((ispalindrome(s,i+1,n-1-i)) || (ispalindrome(s,i,n-i-2)));
        }
        return true;
    }
    public boolean ispalindrome(String s,int start,int end)
    {
        for(int i=start;i<=(start+end)/2;i++)
            if(s.charAt(i)!=s.charAt(end+start-i))
                return false;
        return true;
    }
}
```

---

709. To Lower Case

Implement function ToLowerCase() that has a string parameter str, and returns the same string in lowercase.

Example 1:

```
Input: "Hello"
Output: "hello"
```

Example 2:

```
Input: "here"
Output: "here"
```

Example 3:

```
Input: "LOVELY"
Output: "lovely"
```

https://leetcode.com/problems/to-lower-case/ (https://leetcode.com/problems/to-lower-case/)

```
class Solution {
    public String toLowerCase(String str) {
        char[] lc=str.toCharArray();
        for(int i=0;i<str.length();i++)
        {
            if(lc[i]<=91 && lc[i]>=65)
            lc[i]=(char)(lc[i]+32);
        }
        return String.valueOf(lc);
    }
}
```

771. Jewels and Stones

You're given strings `J` representing the types of stones that are jewels, and `S` representing the stones you have. Each character in `S` is a type of stone you have. You want to know how many of the stones you have are also jewels.

The letters in `J` are guaranteed distinct, and all characters in `J` and `S` are letters. Letters are case sensitive, so `"a"` is considered a different type of stone from `"A"`.

Example 1:

```
Input: J = "aA", S = "aAAbbbb"
Output: 3
```

Example 2:

```
Input: J = "z", S = "ZZ"
Output: 0
```

Note:

- `S` and `J` will consist of letters and have length at most 50.
- The characters in `J` are distinct.

---

https://leetcode.com/problems/jewels-and-stones/ (https://leetcode.com/problems/jewels-and-stones/) easy Problem **My Solution**

```
class Solution {
    public int numJewelsInStones(String J, String S) {
        final int temp=65;
        int freq[]=new int[60];
        char ja[]=J.toCharArray();
        char sa[]=S.toCharArray();
        int cnt=0;
        for(int i=0;i<ja.length;i++)
        {
            freq[ja[i]-temp]++;
        }
        for(int i=0;i<sa.length;i++)
        {
            if(freq[sa[i]-temp]>0)
                cnt++;
        }
        return cnt;
    }
}
```

---

859. Buddy Strings

Given two strings `A` and `B` of lowercase letters, return `true` *if you can swap two letters in* `A` *so the result is equal to* `B`, *otherwise, return* `false`.

Swapping letters is defined as taking two indices `i` and `j` (0-indexed) such that `i != j` and swapping the characters at `A[i]` and `A[j]`. For example, swapping at indices `0` and `2` in `"abcd"` results in `"cbad"`.

Example 1:

```
Input: A = "ab", B = "ba"
Output: true
Explanation: You can swap A[0] = 'a' and A[1] = 'b' to get "ba", which is equal to B.
```

Example 2:

```
Input: A = "ab", B = "ab"
Output: false
Explanation: The only letters you can swap are A[0] = 'a' and A[1] = 'b', which results
```

Example 3:

```
Input: A = "aa", B = "aa"
Output: true
Explanation: You can swap A[0] = 'a' and A[1] = 'a' to get "aa", which is equal to B.
```

Example 4:

```
Input: A = "aaaaaaabc", B = "aaaaaaacb"
Output: true
```

Example 5:

```
Input: A = "", B = "aa"
Output: false
```

Constraints:

- `0 <= A.length <= 20000`
- `0 <= B.length <= 20000`
- `A` and `B` consist of lowercase letters.

https://leetcode.com/problems/buddy-strings/ (https://leetcode.com/problems/buddy-strings/) Explanation: Mainly to handle 3 following coditions :

- If lengths are not equal then return false
- if first character mismatch found then store char of both string in separate variable lets say tempA for string A and tempB for string B then when next mismatch found we have to check whether

  ```
  i) char of A string is equal to tempB ( of string B)
  ii)char of B  string is equal to tempA ( of string A)
  ```

  if Both conditions are true and no other mismatch occurs(flag ==2) then return true
- if both strings are equal(in length as well as value) then we have to check whether any repeating char in string A as we swapping of same char in same string will not change string.
- *MY SOLUTION**

```
class Solution {
   public boolean buddyStrings(String A, String B) {
       if(A.length()!=B.length())
           return false;
       int flag=0;
       boolean ans=false;
       char tempA='A',tempB='A';
       char[] freq=new char[26];
       for(int i=0;i<A.length();i++)
       {
           freq[A.charAt(i)-97]++;
           if(A.charAt(i)!=B.charAt(i))
           {
               if(A.charAt(i)==tempB && B.charAt(i)==tempA && flag==1)
               {
                  ans=true;
               }
               flag++;
               tempB=B.charAt(i);
               tempA=A.charAt(i);
           }
       }
       if(ans==true && flag==2)
           return true;
       else if(flag==0)
       {
           for(int i=0;i<26;i++)
           {
               if(freq[i]>=2)
                   return true;
           }
       }
       return false;
   }
}
```

- *LeetCode Solution**

```
class Solution {
   public boolean buddyStrings(String A, String B) {
       if (A.length() != B.length()) return false;
       if (A.equals(B)) {
           int[] count = new int[26];
           for (int i = 0; i < A.length(); ++i)
               count[A.charAt(i) - 'a']++;

           for (int c: count)
               if (c > 1) return true;
           return false;
       } else {
           int first = -1, second = -1;
           for (int i = 0; i < A.length(); ++i) {
               if (A.charAt(i) != B.charAt(i)) {
                   if (first == -1)
                       first = i;
                   else if (second == -1)
                       second = i;
                   else
                       return false;
               }
           }

           return (second != -1 && A.charAt(first) == B.charAt(second) &&
                   A.charAt(second) == B.charAt(first));
       }
   }
}
```

1108. Defanging an IP Address

Given a valid (IPv4) IP `address`, return a defanged version of that IP address.

A *defanged IP address* replaces every period `"."` with `"[.]"`.

Example 1:

```
Input: address = "1.1.1.1"
Output: "1[.]1[.]1[.]1"
```

Example 2:

```
Input: address = "255.100.50.0"
Output: "255[.]100[.]50[.]0"
```

Constraints:

- The given `address` is a valid IPv4 address.

---

https://leetcode.com/problems/defanging-an-ip-address/ (https://leetcode.com/problems/defanging-an-ip-address/) **My Solution**

```
class Solution {
    public String defangIPaddr(String address) {
        StringBuffer s=new StringBuffer(address+'#');
        int n=s.length();
        int i=0;
        while(s.charAt(i)!='#')
        {
            if(s.charAt(i)=='.')
            {
                s.insert(i,'[');
                s.insert(i+2,']');
                i+=2;
            }
             i++;
        }
        s.deleteCharAt(i);
        return s.toString();
    }
}
```

1221. Split a String in Balanced Strings

Balanced strings are those who have equal quantity of 'L' and 'R' characters.

Given a balanced string `s` split it in the maximum amount of balanced strings.

Return the maximum amount of splitted balanced strings.

Example 1:

```
Input: s = "RLRRLLRLRL"
Output: 4
Explanation: s can be split into "RL", "RRLL", "RL", "RL", each substring contains same
```

Example 2:

```
Input: s = "RLLLLRRRLR"
Output: 3
Explanation: s can be split into "RL", "LLLRRR", "LR", each substring contains same num
```

Example 3:

```
Input: s = "LLLLRRRR"
Output: 1
Explanation: s can be split into "LLLLRRRR".
```

Example 4:

```
Input: s = "RLRRRLLRLL"
Output: 2
Explanation: s can be split into "RL", "RRRLLRLL", since each substring contains an equ
```

Constraints:

- `1 <= s.length <= 1000`
- `s[i] = 'L' or 'R'`

---

https://leetcode.com/problems/split-a-string-in-balanced-strings/ (https://leetcode.com/problems/split-a-string-in-balanced-strings/)

```
class Solution {
    public int balancedStringSplit(String s) {
        int cnt=0,ans=0;
        char [] split=s.toCharArray();
        for(int i=0;i<s.length();i++)
        {
            if(split[i]=='L')
                cnt--;
            else
                cnt++;
            if(cnt==0)
                ans++;
        }
        return ans;
    }
}
```

1324. Print Words Vertically

Given a string `s`. Return all the words vertically in the same order in which they appear in `s`.
Words are returned as a list of strings, complete with spaces when is necessary. (Trailing spaces are not allowed).
Each word would be put on only one column and that in one column there will be only one word.

Example 1:

```
Input: s = "HOW ARE YOU"
Output: ["HAY","ORO","WEU"]
Explanation: Each word is printed vertically.
 "HAY"
 "ORO"
 "WEU"
```

Example 2:

```
Input: s = "TO BE OR NOT TO BE"
Output: ["TBONTB","OEROOE","   T"]
Explanation: Trailing spaces is not allowed.
"TBONTB"
"OEROOE"
"   T"
```

Example 3:

```
Input: s = "CONTEST IS COMING"
Output: ["CIC","OSO","N M","T I","E N","S G","T"]
```

Constraints:

- `1 <= s.length <= 200`
- `s` contains only upper case English letters.
- It's guaranteed that there is only one space between 2 words.

---

https://leetcode.com/problems/print-words-vertically/ (https://leetcode.com/problems/print-words-vertically/) problem no: 1324 /* Explanation: note: Here max_index[] array is only used to not use inbuilt trim function Algorithm:

1) find largest length word (max_count) in given string which will be number of words in final answer 2) then iterate over all words of string for max_count times and add each letter of word to max_count words in ans with some condition / *My solution**

```
class Solution {
    public List<String> printVertically(String s) {
        List <String> word =new ArrayList<String>();
        StringBuilder temp=new StringBuilder();
        int max_count=0;
        for(int i=0;i<s.length();i++)
        {
            if(s.charAt(i)!=' ')
                temp.append(s.charAt(i));
            else
            {
                word.add(temp.toString());
                max_count=Math.max(max_count,temp.length());
                temp=new StringBuilder();
            }
        }
                word.add(temp.toString());
                max_count=Math.max(max_count,temp.length());
                temp=new StringBuilder();

        int max_index[]=new int[word.size()];
        max_index[word.size()-1]=word.get(word.size()-1).length();
        for(int i=word.size()-2;i>=0;i--)
        {
            max_index[i]=Math.max(max_index[i+1],word.get(i).length());
        }

        List <String> ans=new ArrayList<String>();
        for(int i=0;i<max_count;i++)
        {
            temp=new StringBuilder();
            for(int j=0;j<word.size();j++)
            {
                if(i<word.get(j).length())
                    temp.append(word.get(j).charAt(i));
                else if(ans.size()<max_index[j])
                    temp.append(" ");
            }
            ans.add(temp.toString());
        }
        return ans;
    }
}
```

without max_index and using Righttrim()

```
String[]str=s.split(" ");
    List<String>list=new ArrayList<>();
    int max=0;
    for(String st:str){
        max=Math.max(max,st.length());
    }
    for(int i=0;i<max;i++){
        StringBuilder sb=new StringBuilder();
        for(String st:str){
            if(i<st.length()){
                sb.append(st.charAt(i));
            }else{
                sb.append(" ");
            }
        }
        list.add(sb.toString().stripTrailing());
    }
    return list;
```

1347. Minimum Number of Steps to Make Two Strings Anagram

Given two equal-size strings `s` and `t`. In one step you can choose **any character** of `t` and replace it with **another character**.

Return *the minimum number of steps* to make `t` an anagram of `s`.

An **Anagram** of a string is a string that contains the same characters with a different (or the same) ordering.

Example 1:

```
Input: s = "bab", t = "aba"
Output: 1
Explanation: Replace the first 'a' in t with b, t = "bba" which is anagram of s.
```

Example 2:

```
Input: s = "leetcode", t = "practice"
Output: 5
Explanation: Replace 'p', 'r', 'a', 'i' and 'c' from t with proper characters to make t
```

Example 3:

```
Input: s = "anagram", t = "mangaar"
Output: 0
Explanation: "anagram" and "mangaar" are anagrams.
```

Example 4:

```
Input: s = "xxyyzz", t = "xxyyzz"
Output: 0
```

Example 5:

```
Input: s = "friend", t = "family"
Output: 4
```

Constraints:

- `1 <= s.length <= 50000`
- `s.length == t.length`
- `s` and `t` contain lower-case English letters only.

---

https://leetcode.com/problems/minimum-number-of-steps-to-make-two-strings-anagram/ (https://leetcode.com/problems/minimum-number-of-steps-to-make-two-strings-anagram/) **My Solution** Using Hashmap Or Frequency Array Increment if char found in s Decrement if char found in t consider either values less than 0 or values greater than 0 or add all absolute value in frequency array and add divide by 2;

```
class Solution {
    public int minSteps(String s, String t) {
        int freq[]=new int[26];
        int cnt=0;
        for(int i=0;i<s.length();i++)
        {
            freq[s.charAt(i)-'a']++;
            freq[t.charAt(i)-'a']--;
        }
        for(int i=0;i<26;i++)
        {
            if(freq[i]>0)
            cnt+=freq[i];
        }
        return cnt;
    }
}
```

```
class Solution {
 public int minSteps(String s, String t) {
     if(s.equals(t))return 0; //removed this as it is already given length of two str
ings are same! silly mistake

     int n = s.length(), ans = 0;
     int[] arr = new int[26];
     char[] sc=s.toCharArray();   //converted string to char array
     char[] tc=t.toCharArray();   //converted string to char array
     for(int i = 0; i < n; i++) {
         arr[sc[i] - 'a']++;
         arr[tc[i] - 'a']--;
     }
     for(int i : arr) if(i > 0) ans += i;
     return ans;
 }
}
```

IMPROVS-CONVERTING STRING TO CHAR ARRAY AS ARRAY IS FASTER ! RUNTIME -9 MS!

REPLACE SOME CONSTANTS WITTH FINAL STATIC VARIABLES! INSTEAD OF EVERYTIME CALCULATING THE ASCII OF a WE ARE REPLACING IT WITH A CONSTANT! THIS REDUCES RUNTIME IMMENSELY!

ATTEMPT -3

class Solution { final static int a = 97; //ASCII OF 'a' final static int size = 26; //size of array

```
    public int minSteps(String s, String t) {
        int[] arr = new int[size];
        int l = s.length();
        char[] sa = s.toCharArray();
        char[] ta = t.toCharArray();
        for (int i = 0; i < l; i++) {
            int sci = sa[i] - a;
            int tci = ta[i] - a;
            arr[sci] += 1;
            arr[tci] -= 1;
        }
        int ans = 0;
        for (int n : arr) {
            if (n > 0) {
                ans += n;
            }
        }
        return ans;
    }
}
```

1370. Increasing Decreasing String

Given a string `s`. You should re-order the string using the following algorithm:

1. Pick the **smallest** character from `s` and **append** it to the result.
2. Pick the **smallest** character from `s` which is greater than the last appended character to the result and **append** it.
3. Repeat step 2 until you cannot pick more characters.
4. Pick the **largest** character from `s` and **append** it to the result.
5. Pick the **largest** character from `s` which is smaller than the last appended character to the result and **append** it.
6. Repeat step 5 until you cannot pick more characters.
7. Repeat the steps from 1 to 6 until you pick all characters from `s`.

In each step, If the smallest or the largest character appears more than once you can choose any occurrence and append it to the result.

Return *the result string* after sorting `s` with this algorithm.

Example 1:

```
Input: s = "aaaabbbbcccc"
Output: "abccbaabccba"
Explanation: After steps 1, 2 and 3 of the first iteration, result = "abc"
After steps 4, 5 and 6 of the first iteration, result = "abccba"
First iteration is done. Now s = "aabbcc" and we go back to step 1
After steps 1, 2 and 3 of the second iteration, result = "abccbaabc"
After steps 4, 5 and 6 of the second iteration, result = "abccbaabccba"
```

Example 2:

```
Input: s = "rat"
Output: "art"
Explanation: The word "rat" becomes "art" after re-ordering it with the mentioned algor
```

Example 3:

```
Input: s = "leetcode"
Output: "cdelotee"
```

Example 4:

```
Input: s = "ggggggg"
Output: "ggggggg"
```

Example 5:

```
Input: s = "spo"
Output: "ops"
```

Constraints:

- `1 <= s.length <= 500`
- `s` contains only lower-case English letters.

---

https://leetcode.com/problems/increasing-decreasing-string/ (https://leetcode.com/problems/increasing-decreasing-string/) problem no 1370 /* Explanation:

1) store each char of string in frequency array 2) loop till length of ans is not equal to length of string and do following 3) two loops one from 0 to 25 and other from 25 to 0 4) if we found char present then append to answer and decrement freq array val / *Solution**

```
class Solution {
    public String sortString(String s) {
        int frequency[]=new int[26];
        for(int i=0;i<s.length();i++)
        {
            frequency[s.charAt(i)-'a']++;
        }
        StringBuilder ans=new StringBuilder();
        while(ans.length()!=s.length())
        {
            for(int i=0;i<26;i++)
            {
                if(frequency[i]>0)
                {
                    ans.append((char)(i+97));
                    frequency[i]--;
                }
            }
            for(int i=25;i>=0;i--)
            {
                if(frequency[i]>0)
                {
                    ans.append((char)(i+97));
                    frequency[i]--;
                }
            }
        }
        return ans.toString();
    }
}
```

Solution with one loop pass only

```
class Solution {
    public String sortString(String s) {
        int frequency[]=new int[26];
        for(int i=0;i<s.length();i++)
        {
            frequency[s.charAt(i)-'a']++;
        }
        StringBuilder ans=new StringBuilder();
        StringBuilder ans1;
        StringBuilder ans2;
        while(ans.length()!=s.length())
        {
            ans1=new StringBuilder();
            ans2=new StringBuilder();
            for(int i=0;i<26;i++)
            {
                if(frequency[i]>0)
                {
                    ans1.append((char)(i+97));
                    frequency[i]--;
                }
//This condition ensures that if letter between M and z should be 2 or more as they a
re not covered in ans1
//but need not to be true and required freq 1 or more for letter from A to M as thy a
re already cover in asn1
                if((frequency[25-i]>1 && 25-i>=12) || (frequency[25-i]>0 && 25-i<=1
2))
                {
                    ans2.append((char)(122-i));
                    frequency[25-i]--;
                }
            }
            ans.append(ans1.toString());
            ans.append(ans2.toString());
        }
        return ans.toString();
    }
}
```

1374. Generate a String With Characters That Have Odd Counts

Given an integer n , *return a string with n characters such that each character in such string occurs* ***an odd number of times***.

The returned string must contain only lowercase English letters. If there are multiples valid strings, return **any** of them.

Example 1:

```
Input: n = 4
Output: "pppz"
Explanation: "pppz" is a valid string since the character 'p' occurs three times and th
```

Example 2:

```
Input: n = 2
Output: "xy"
Explanation: "xy" is a valid string since the characters 'x' and 'y' occur once. Note t
```

Example 3:

```
Input: n = 7
Output: "holasss"
```

Constraints:

- `1 <= n <= 500`

---

https://leetcode.com/problems/generate-a-string-with-characters-that-have-odd-counts/ (https://leetcode.com/problems/generate-a-string-with-characters-that-have-odd-counts/) if n is odd then all n "aaaaa...a' here length of a is odd if n is even then all n "aaaaa...b" here length of a is odd ,length of b is odd

```
class Solution {
    public String generateTheString(int n) {
        StringBuffer s=new StringBuffer();
        for(int i=0;i<n-1;i++)
        {
            s.append('a');
        }
        if((n & 1)==0)
            s.append('b');
        else
            s.append('a');
        return s.toString();
    }
}
```

1487. Making File Names Unique

Given an array of strings `names` of size `n`. You will create `n` folders in your file system **such that**, at the `ith` minute, you will create a folder with the name `names[i]`.

Since two files **cannot** have the same name, if you enter a folder name which is previously used, the system will have a suffix addition to its name in the form of `(k)`, where, `k` is the **smallest positive integer** such that the obtained name remains unique.

Return *an array of strings of length* `n` where `ans[i]` is the actual name the system will assign to the `ith` folder when you create it.

Example 1:

```
Input: names = ["pes","fifa","gta","pes(2019)"]
Output: ["pes","fifa","gta","pes(2019)"]
Explanation: Let's see how the file system creates folder names:
"pes" --> not assigned before, remains "pes"
"fifa" --> not assigned before, remains "fifa"
"gta" --> not assigned before, remains "gta"
"pes(2019)" --> not assigned before, remains "pes(2019)"
```

Example 2:

```
Input: names = ["gta","gta(1)","gta","avalon"]
Output: ["gta","gta(1)","gta(2)","avalon"]
Explanation: Let's see how the file system creates folder names:
"gta" --> not assigned before, remains "gta"
"gta(1)" --> not assigned before, remains "gta(1)"
"gta" --> the name is reserved, system adds (k), since "gta(1)" is also reserved, syste
"avalon" --> not assigned before, remains "avalon"
```

Example 3:

```
Input: names = ["onepiece","onepiece(1)","onepiece(2)","onepiece(3)","onepiece"]
Output: ["onepiece","onepiece(1)","onepiece(2)","onepiece(3)","onepiece(4)"]
Explanation: When the last folder is created, the smallest positive valid k is 4, and i
```

Example 4:

```
Input: names = ["wano","wano","wano","wano"]
Output: ["wano","wano(1)","wano(2)","wano(3)"]
Explanation: Just increase the value of k each time you create folder "wano".
```

Example 5:

```
Input: names = ["kaido","kaido(1)","kaido","kaido(1)"]
Output: ["kaido","kaido(1)","kaido(2)","kaido(1)(1)"]
Explanation: Please note that system adds the suffix (k) to current name even it contai
```

Constraints:

- `1 <= names.length <= 5 * 10^4`
- `1 <= names[i].length <= 20`
- `names[i]` consists of lower case English letters, digits and/or round brackets.

---

https://leetcode.com/problems/making-file-names-unique/ (https://leetcode.com/problems/making-file-names-unique/) problem no 1487 /* Explanation:

1) create hashmap for all given string containing in given list and integer 2) Integer is initially 1 which indicate version of that file it will increse if same version is found / *My solution**

```
class Solution {
    public String[] getFolderNames(String[] names) {
        HashMap <String,Integer> h=new HashMap <String,Integer>();
        for(int i=0;i<names.length;i++)
        {
            if(h.containsKey(names[i]))
            {
                int counter=h.get(names[i]);
                StringBuilder s=new StringBuilder(names[i]);
                s.append('(');
                s.append(counter);
                s.append(')');
                counter++;
                while(h.containsKey(s.toString()))
                {
                    s=new StringBuilder(names[i]);
                    s.append('(');
                    s.append(counter);
                    s.append(')');
                    counter++;
                }
                h.put(s.toString(),1);
                h.put(names[i],counter);
                names[i]=s.toString();
            }
            else
            {
                h.put(names[i],1);
            }
        }
        return names;
    }
}
```

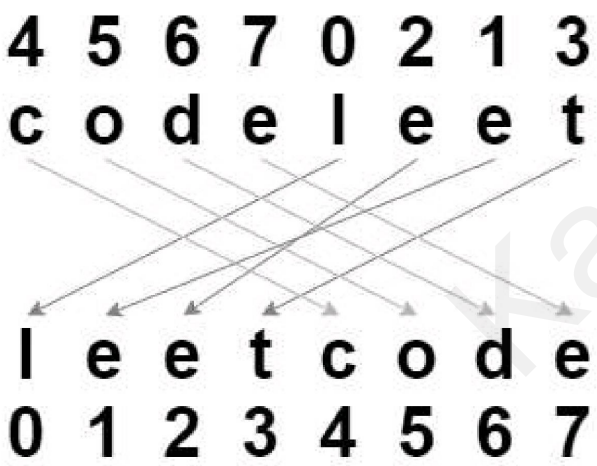
1528. Shuffle String

Given a string `s` and an integer array `indices` of the **same length**.

The string `s` will be shuffled such that the character at the i^{th} position moves to `indices[i]` in the shuffled string.

Return *the shuffled string*.

Example 1:




```
Input: s = "codeleet", indices = [4,5,6,7,0,2,1,3]
Output: "leetcode"
Explanation: As shown, "codeleet" becomes "leetcode" after shuffling.
```

Example 2:

```
Input: s = "abc", indices = [0,1,2]
Output: "abc"
Explanation: After shuffling, each character remains in its position.
```

Example 3:

```
Input: s = "aiohn", indices = [3,1,4,2,0]
Output: "nihao"
```

Example 4:

```
Input: s = "aaiougrt", indices = [4,0,2,6,7,3,1,5]
Output: "arigatou"
```

Example 5:

```
Input: s = "art", indices = [1,0,2]
Output: "rat"
```

Constraints:

- `s.length == indices.length == n`
- `1 <= n <= 100`
- `s` contains only lower-case English letters.
- `0 <= indices[i] < n`
- All values of `indices` are unique (i.e. `indices` is a permutation of the integers from `0` to `n - 1`).

https://leetcode.com/problems/shuffle-string/ (https://leetcode.com/problems/shuffle-string/) **My solution**
O(n) time complexity O(n) space complexity

```
class Solution {
    public String restoreString(String s, int[] indices) {
        char []ans=new char[s.length()];
        for(int i=0;i<indices.length;i++)
        {
            ans[indices[i]]=s.charAt(i);
        }
        return String.valueOf(ans);
    }
}
```