class Solution {

    public void merge(int[] nums1, int m, int[] nums2, int n) {

        int k = m + n - 1; // Pointer for the last position in nums1

        int i = m - 1; // Pointer for the last element in the initial part of nums1

        int j = n - 1; // Pointer for the last element in nums2

       while(j>=0)

       {

         if(i>=0 && nums1[i]>nums2[j])

         {

            nums1[k--]=nums1[i--];

         }

         else

         {

             nums1[k--]=nums2[j--];

         }

       }

    }

}

class Solution {

    public int removeElement(int[] nums, int val) {

       int index=0;

       for(int i=0;i<nums.length;i++)

       {

        if(nums[i]!=val)

        {

            nums[index]=nums[i];

            index++;

        }

       }

       return index;

    }

}

class Solution {

    public int maxProfit(int[] prices) {

        int buy = prices[0];

        int profit = 0;

        for (int i = 1; i < prices.length; i++) {

            if (prices[i] < buy) {

                buy = prices[i];

            } else if (prices[i] - buy > profit) {

                profit = prices[i] - buy;

            }

        }

        return profit;

    }

}

[290. Word Pattern](https://leetcode.com/problems/word-pattern/)

import java.util.HashMap;

import java.util.Map;

class Solution {

    public boolean wordPattern(String pattern, String s) {

        Map<Character, String> map = new HashMap<>();

        String[] arr = s.split(" ");

        if (arr.length != pattern.length()) {

            return false;

        }

        for (int i = 0; i < pattern.length(); i++) {

            char c = pattern.charAt(i);

            String word = arr[i];

            if (map.containsKey(c)) //checks if the character c from the pattern is already a key in the map.

            {

                if (!map.get(c).equals(word))

                //map.get(c) retrieves the word that c is currently mapped to.

//!map.get(c).equals(word) checks if this word is different from the current word in the string arr[i].

                {

                    return false;

                }

            } else

            //map.containsValue(word) checks if the word word is already mapped to another character.

//If word is found in the map values, it means this word is already associated with a different character, hence return false as this ////violates the one-to-one mapping rule. {

                if (map.containsValue(word)) {

                    return false;

                }

                map.put(c, word);//If the word is not yet mapped to any character, map.put(c, word) adds the new mapping of character c to the word word.

            }

        }

        return true;

    }

}

[Best Time to Buy and Sell Stock](https://leetcode.com/problems/best-time-to-buy-and-sell-stock/)class Solution {

    public int maxProfit(int[] prices) {

        int buy = prices[0];

        int profit = 0;

        for (int i = 1; i < prices.length; i++) {

            if (prices[i] < buy) {

                buy = prices[i];

            } else if (prices[i] - buy > profit) {

                profit = prices[i] - buy;

            }

        }

        return profit;

    }

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public class LongestPalindromeSubstring {

public static String longestPalindrome(String s) {

if (s == null || s.length() < 1) return "";

String longestPalindrome = "";

int maxLength = 0;

for (int i = 0; i < s.length(); i++) {

for (int j = i + 1; j <= s.length(); j++) {

String substr = s.substring(i, j);

if (isPalindrome(substr) && substr.length() > maxLength) {

maxLength = substr.length();

longestPalindrome = substr;

}

}

}

return longestPalindrome;

}

private static boolean isPalindrome(String s) {

StringBuilder sb = new StringBuilder(s);

return sb.reverse().toString().equals(s);

}

public static void main(String[] args) {

String inputString = "babad";

String result = longestPalindrome(inputString);

System.out.println("Longest palindromic substring: " + result);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*import java.util.Scanner;

public class ReverseWord {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

String input;

System.out.println("Enter a String");

input=sc.nextLine();

String output=reverseStringArray(input);

System.out.println(output);

}

private static String reverseStringArray(String input) {

String[] word = input.trim().split("\\s+");

StringBuilder sb=new StringBuilder();

for(int i=word.length-1;i>=0;i--)

{

sb.append(word[i]);

if(i!=0)

{

sb.append(" ");

}

}

return sb.toString();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/import java.util.HashMap;

import java.util.Scanner;

public class ReverseWord {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a String:");

String input = sc.nextLine();

String output = reverseWords(input);

System.out.println(output);

}

private static String reverseWords(String input) {

// Step 1: Trim leading/trailing whitespace and split the string by one or more spaces

String[] words = input.trim().split("\\s+");

// Step 2: Use a HashMap to store the reversed order of words

HashMap<Integer, String> map = new HashMap<>();

for (int i = 0; i < words.length; i++) {

map.put(words.length - 1 - i, words[i]);

}

// Step 3: Construct the reversed string

StringBuilder sb = new StringBuilder();

for (int i = 0; i < words.length; i++) {

sb.append(map.get(i));

if (i != words.length - 1) { // Append a space after each word except the last one

sb.append(" ");

}

}

return sb.toString();

}

}

Given an array nums of size n, return the majority element.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

class Solution {

    public int majorityElement(int[] nums) {

       int count=0,candidate=nums[0];

       for(int i=0;i<nums.length;i++)

       {

        if(count==0)

        {

            candidate=nums[i];

        }

        else

        {

            if(nums[i]==candidate)

            {

                count++;

            }

            else

            {

                count--;

            }

        }

       }

       return candidate;

    }

}

ISOMORPHIC STRING

//One To One Mapping

class Solution {

    public boolean isIsomorphic(String s, String t) {

     Map<Character, Character> map = new HashMap<>();

        for (int i = 0; i < s.length(); i++)

        {

            char sChar = s.charAt(i);

            char tChar = t.charAt(i);

            if (map.containsKey(sChar))

             {

                if (map.get(sChar) != tChar) {

                    return false;  // sChar is mapped to a different character

                      }

             }

              else {

                if (map.containsValue(tChar))

{

                    return false;  // tChar is already mapped to another sChar

                }

                map.put(sChar, tChar);  // map sChar to tChar

            }

        }

        return true;

    }

}

[58. Length of Last Word](https://leetcode.com/problems/length-of-last-word/)

class Solution {

    public int lengthOfLastWord(String s) {

        String arr[]=s.trim().split("\\s+");

        String last=arr[arr.length-1];

         int length=last.length();

         return length;

    }

}

public class rotate {

public static void main(String[] args) {

rotate r=new rotate();

int a[]= {1,2,3,4,5,6,7};

r.rotate(a, 3);

}

public void rotate(int[] nums, int k) {

k %= nums.length;

reverse(nums, 0, nums.length-1);//reverse whole array

reverse(nums, 0, k-1);//reverse before mid =k

reverse(nums, k, nums.length-1);//reverse after mid=k

for(int i=0;i<nums.length;i++)

{

System.***out***.println(nums[i]);

}

}

public void reverse(int[] nums, int start, int end){

while(start < end){

int temp = nums[start];

nums[start] = nums[end];

nums[end] = temp;

start++;

end--;

}

}

}

JUMP GAME

class Solution {

    public boolean canJump(int[] nums) {

        int n=nums.length;

        int reachable=0;

        for(int i=0;i<n;i++)

        {

            if(reachable < i)

               return false;

            reachable = Math.max(reachable,i+nums[i]);

        }

    return true;

}

}

**[380. Insert Delete GetRandom O(1)](https://leetcode.com/problems/insert-delete-getrandom-o1/)**

class RandomizedSet {

    List<Integer> list;

    HashMap<Integer,Integer> map;

    Random random =new Random();

    public RandomizedSet() {

        list=new ArrayList<Integer>();

        map=new HashMap<Integer,Integer>();

    }

    public boolean insert(int val) {

        if(map.containsKey(val))

        {

            return false;

        }

        map.put(val,list.size());

        list.add(val);

        return true;

    }

    public boolean remove(int val) {

         if(!map.containsKey(val))

        {

            return false;

        }

        int pos=map.get(val);

        if(pos!=(list.size()-1))

        {

            int lastElement=list.get(list.size()-1);

            list.set(pos,lastElement);

            map.put(lastElement,pos);

        }

        map.remove(val);

        list.remove(list.size()-1);

        return true;

    }

    public int getRandom() {

        int randomInt=random.nextInt(list.size());

        return list.get(randomInt);

    }

}

/\*\*

 \* Your RandomizedSet object will be instantiated and called as such:

 \* RandomizedSet obj = new RandomizedSet();

 \* boolean param\_1 = obj.insert(val);

 \* boolean param\_2 = obj.remove(val);

 \* int param\_3 = obj.getRandom();

 \*/

**[238. Product of Array Except Self](https://leetcode.com/problems/product-of-array-except-self/)**

class Solution {

    public int[] productExceptSelf(int[] nums) {

        int leftProduct[] = new int[nums.length];

        int rightProduct[] = new int[nums.length];

        // Calculate left product

        for (int i = 0, temp = 1; i < nums.length; i++) {

            leftProduct[i] = temp;

            temp \*= nums[i];

        }

        // Calculate right product

        for (int i = nums.length - 1, temp = 1; i >= 0; i--) {

            rightProduct[i] = temp;

            temp \*= nums[i];

        }

        int result[] = new int[nums.length];

        for (int i = 0; i < nums.length; i++) {

            result[i] = leftProduct[i] \* rightProduct[i];

        }

        return result;

    }

}

Code

Testcase

Testcase

Test Result

**[134. Gas Station](https://leetcode.com/problems/gas-station/)**

class Solution {

    public int canCompleteCircuit(int[] gas, int[] cost) {

            int totalDiff=0,fuel=0,index=0;

        for(int i=0;i<gas.length;i++)

        {

            int diff=gas[i]-cost[i];

            totalDiff=totalDiff+diff;

            fuel=fuel+diff;

            if(fuel<0)

            {

                index=i+1;

                fuel=0;

            }

        }

        if(totalDiff<0)

        {

            return -1;

        }

        else

        {

            return index;

        }

    }

}

**[68. Text Justification](https://leetcode.com/problems/text-justification/)**

class Solution {

    public List<String> fullJustify(String[] words, int maxWidth) {

        List<String> ans = new ArrayList<>();

        int i = 0;

        while (i < words.length) {

            StringBuilder sb = new StringBuilder();

            while (i < words.length && sb.length() + words[i].length() <= maxWidth) {

                sb.append(words[i]).append(" ");

                i++;

            }

            String str = sb.toString().trim();

            if (i < words.length)

                ans.add(addSpaces(str, maxWidth));

            else {

                while (str.length() != maxWidth)

                    str += " ";

                ans.add(str);

            }

        }

        return ans;

    }

    private String addSpaces(String s, int width) {

        StringBuilder sb = new StringBuilder();

        String[] arr = s.split(" ");

        int count = arr.length - 1;

        if (count == 0) {

            while (s.length() != width)

                s += " ";

            return s;

        }

        int padding = width - s.length();

        int equalSpace = padding / count;

        int moreNeeded = padding % count;

        String spaces = " ";

        while (equalSpace-- > 0)

            spaces += " ";

        String sp = spaces + " ";

        for (String ele : arr) {

            if (moreNeeded-- > 0)

                sb.append(ele + sp);

            else

                sb.append(ele + spaces);

        }

        return sb.toString().trim();

    }

}

e

Testcase

Testcase

Test Result

**[383. Ransom Note](https://leetcode.com/problems/ransom-note/)**

class Solution {

    public boolean canConstruct(String ransomNote, String magazine) {

        int count=0;

        Map<Character,Integer> map=new HashMap<>();

        for(int i=0;i<magazine.length();i++)

        {

            if(map.containsKey(magazine.charAt(i)))

            {

                map.put(magazine.charAt(i),map.get(magazine.charAt(i))+1);

            }

            else

            {

             map.put(magazine.charAt(i),1);

            }

        }

        for(int i=0;i<ransomNote.length();i++)

        {

            char ch=ransomNote.charAt(i);

            if(map.containsKey(ch) && map.get(ch)>0)

            {

                count++;

                map.put(ch,map.get(ch)-1);

            }

            else

            {

                break;

            }

        }

        if(count==ransomNote.length())

        {

            return true;

        }

        else

        {

               return false;

        }

    }

}