1. Two Sum

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to* target.

You may assume that each input would have *exactly* one solution, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

Only one valid answer exists.

```
public int[] TwoSum(int[] nums, int target)
        {
            Dictionary<int, int> map = new Dictionary<int, int>();
            for(int i=0;i<nums.Length;i++)
            {
                int curr = nums[i];
                int x = target - curr;
                if (map.ContainsKey(x))
                    return new int[] { map[x], i };
                if(!map.ContainsKey(curr))
                    map.Add(curr, i);
            }
            return null;
        }

TC: O(n)
SC: O(n)
```

2. 3Sum

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

Example 1:

```
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not matter.
```

Example 2:

```
Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.
```

Example 3:

```
Input: nums = [0,0,0]
Output: [[0,0,0]]
Explanation: The only possible triplet sums up to 0.
```

```
public IList<IList<int>> ThreeSum(int[] nums)
        {
            Array.Sort(nums);
            IList<IList<int>> result = new List<IList<int>>();
            for (int i = 0; i < nums.Length && nums[i] <= 0; i++)
            {
                if (i == 0 || nums[i] != nums[i - 1])
                    TwoSum(nums, i, result);

            }
            return result;
        }

        private void TwoSum(int []nums, int i, IList<IList<int>> result)
        {
            int l = i+1;
            int r = nums.Length - 1;

            while(l<r)
            {
                int sum = nums[i] + nums[l] + nums[r];
                if (sum < 0)
                    l++;
                else if (sum > 0)
                    r--;
                else
                {
                    result.Add(new List<int> { nums[i], nums[l], nums[r] });
                    l++;
                    r--;
                    while (l < r && nums[l] == nums[l - 1])
                        ++l;
                }
            }
        }
```

TC: O(n^2)
SC: O(1)

3. 3Sum Closest

Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target. Return *the sum of the three integers*.

You may assume that each input would have exactly one solution.

Example 1:

```
Input: nums = [-1,2,1,-4], target = 1
Output: 2
Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
```

Example 2:

```
Input: nums = [0,0,0], target = 1
Output: 0
```

```
Explanation: The sum that is closest to the target is 0. (0 + 0 + 0 = 0).
```

```
public int ThreeSumClosest(int[] nums, int target)
        {
            Array.Sort(nums);

            int closestValue = int.MaxValue;

            for (int i = 0; i<nums.Length && closestValue!=0; i++)
            {
                TwoSum(nums, i, target, ref closestValue);
            }
            return target - closestValue;
        }

        private void TwoSum(int[] nums, int i, int target, ref int closestValue)
        {
            int l = i + 1;
            int r = nums.Length - 1;
            while (l<r)
            {
                int sum = nums[i] + nums[l] + nums[r];

                if (Math.Abs(closestValue) > Math.Abs(sum - target))
                    closestValue = target - sum;

                if (sum < target)
                    l++;
                else
                    r--;
            }
        }
```

TC: O(n^2)
SC: O(1)

4. Longest Substring Without Repeating Characters

Given a string s, find the length of the longest substring without repeating characters.

Example 1:

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.
```

```
public int LengthOfLongestSubstring(string s)
        {
            int n = s.Length;

            int i=0;
            int ans = 0;
            Dictionary<char, int> map = new Dictionary<char, int>();

            for(int j=0; j<n; j++)
            {
                if(map.ContainsKey(s[j]))
                {
                    i = Math.Max(i, map[s[j]]);
                    map[s[j]] = j+1;
                }
                else
                    map.Add(s[j], j+1);
                ans = Math.Max(ans, j-i+1);
            }

            return ans;
        }

TC: O(n)
SC: O(min(m,n))
m -> size of charset
```

5. Longest Palindromic Substring

Given a string `s`, return *the longest palindromic substring* in `s`.

A string is called a palindrome string if the reverse of that string is the same as the original string.

Example 1:

```
Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.
```

Example 2:

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

```
public string LongestPalindrome(string s)
        {
            int n = s.Length;
if (n < 2)
                return s;
            bool[,] table = new bool[n,n];
            int start = 0;
            int maxLength = 1;
            for(int i=0;i<n;i++)
            {
                table[i, i] = true;
            }

            for (int i = 0; i < n-1; i++)
            {
                if (s[i] == s[i + 1])
                {
                    table[i, i + 1] = true;
                    start = i;
                    maxLength = 2;
                }
            }

            for(int k=3;k<=n;k++)
            {
                for(int i=0;i<n-k+1;i++)
                {
                    int j = i + k - 1;

                    if (s[i] == s[j] && table[i + 1, j - 1])
                    {
                        table[i, j] = true;
                        if (k > maxLength)
                        {
                            start = i;
                            maxLength = k;
                        }
                    }
                }
            }
            return s.Substring(start, maxLength);
        }
```

TC: O(n^2)

SC: O(n^2)

6. String to Integer (atoi)

Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function).

The algorithm for myAtoi(string s) is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range [-231, 231 - 1], then clamp the integer so that it remains in the range. Specifically, integers less than -231 should be clamped to -231, and integers greater than 231 - 1 should be clamped to 231 - 1.
6. Return the integer as the final result.

Note:

- Only the space character ' ' is considered a whitespace character.
- Do not ignore any characters other than the leading whitespace or the rest of the string after the digits.

Example 1:

```
Input: s = "42"
Output: 42
Explanation: The underlined characters are what is read in, the caret is the current reader position.
Step 1: "42" (no characters read because there is no leading whitespace)
         ^
Step 2: "42" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "42" ("42" is read in)
           ^
The parsed integer is 42.
Since 42 is in the range [-231, 231 - 1], the final result is 42.
```

Example 2:

```
Input: s = "   -42"
Output: -42
Explanation:
Step 1: "   -42" (leading whitespace is read and ignored)
            ^
Step 2: "   -42" ('-' is read, so the result should be negative)
             ^
Step 3: "   -42" ("42" is read in)
               ^
The parsed integer is -42.
Since -42 is in the range [-231, 231 - 1], the final result is -42.
```

Example 3:

```
Input: s = "4193 with words"
Output: 4193
Explanation:
Step 1: "4193 with words" (no characters read because there is no leading whitespace)
         ^
Step 2: "4193 with words" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "4193 with words" ("4193" is read in; reading stops because the next character is a non-digit)
             ^
The parsed integer is 4193.
Since 4193 is in the range [-231, 231 - 1], the final result is 4193.
```

```
public int MyAtoi(string s)
        {
            int sign = 1;
            long result = 0;
            bool digitFound = false;
            bool isSignFound = false;
            foreach(char c in s)
            {
                if (digitFound == true && !char.IsDigit(c))
                    break;
                if (c == ' ')
                {
                    if (isSignFound)
                        return 0;
                    continue;
                }
                else if (c == '-')
                {
                    if (isSignFound)
                        return 0;
                    sign = -1;
                    isSignFound = true;
                }
                else if (c == '+')
                {
                    if (isSignFound)
                        return 0;
                    sign = 1;
                    isSignFound = true;
                }
                else if (char.IsDigit(c))
                {
                    int currentDigit = int.Parse(c.ToString());
                    digitFound = true;
                    if (result == 0)
                        result = currentDigit;
                    else
                    {
                        result = result * 10 + currentDigit;

                        if (result * sign < int.MinValue)
                            return int.MinValue;
                        if (result * sign > int.MaxValue)
                            return int.MaxValue;
                    }
                }
                else
                    return (int)result;
            }
            return (int)(result * sign);
        }

TC: O(n)
SC: O(1)
```

7. Container With Most Water

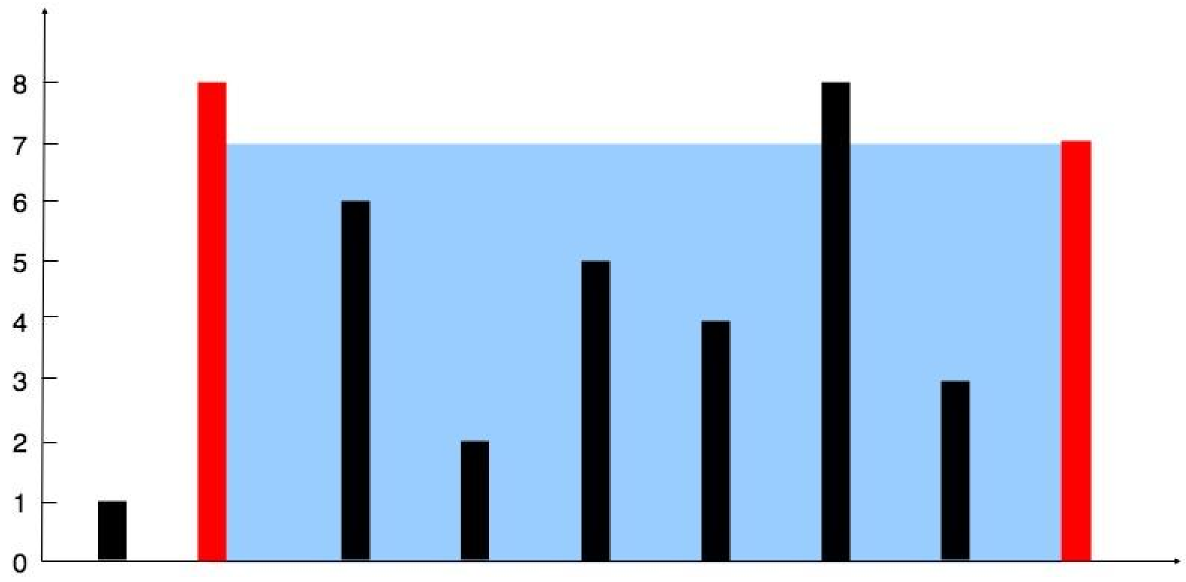
You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water

Return *the maximum amount of water a container can store*.

Notice that you may not slant the container.

Example 1:



```
Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water
(blue section) the container can contain is 49.
```

Example 2:

```
Input: height = [1,1]
Output: 1
```

```
public int MaxArea(int[] height)
        {
            int length = height.Length;
            int maxArea = 0;
            int left = 0;
            int right = length - 1;
            while(left < right)
            {
                maxArea = Math.Max(maxArea, Math.Min(height[left], height[right]) * (right -
left));
                if (height[left] < height[right])
                    left++;
                else
                    right--;
            }
            return maxArea;
        }

TC: O(n)
SC: O(1)
```

8. Integer to Roman

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, 2 is written as II in Roman numeral, just two one's added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

Given an integer, convert it to a roman numeral.

Example 1:

```
Input: num = 3
Output: "III"
Explanation: 3 is represented as 3 ones.
```

Example 2:

```
Input: num = 58
Output: "LVIII"
Explanation: L = 50, V = 5, III = 3.
```

Example 3:

```
Input: num = 1994
Output: "MCMXCIV"
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
```

```
public string IntToRoman(int num)
        {
            StringBuilder result = new StringBuilder();
            while(num!=0)
            {
                if(num>=1000)
                {
                    result.Append("M");
                    num = num - 1000;
                }
                else if(num>=900)
                {
                    result.Append("CM");
                    num = num - 900;
                }
                else if(num>=500)
                {
                    result.Append("D");
                    num = num - 500;
                }
                else if(num>=400)
                {
                    result.Append("CD");
                    num = num - 400;
                }
                else if(num>= 100)
                {
                    result.Append("C");
                    num = num - 100;
                }
                else if(num>=90)
                {
                    result.Append("XC");
                    num = num - 90;
                }
                else if(num>=50)
                {
                    result.Append("L");
                    num = num - 50;
                }
                else if(num>=40)
                {
                    result.Append("XL");
                    num = num - 40;
                }
                else if (num >= 10)
                {
                    result.Append("X");
                    num = num - 10;
                }
                else if (num >= 9)
                {
                    result.Append("IX");
                    num = num - 9;
                }
                else if (num >= 5)
                {
                    result.Append("V");
                    num = num - 5;
                }
                else if (num >= 4)
                {
                    result.Append("IV");
                    num = num - 4;
                }
                else
                {
                    result.Append("I");
                    num = num - 1;
                }
            }
            return result.ToString();
        }

TC: O(n)
SC: O(1)
```

9. Roman to Integer

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Example 1:

```
Input: s = "III"
Output: 3
Explanation: III = 3.
```

Example 2:

```
Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.
```

Example 3:

```
Input: s = "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
```

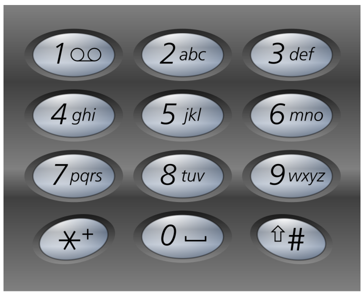
```
public int RomanToInt(string s) {
        Dictionary<string, int> romanMapping = new Dictionary<string, int>
            {
                { "M", 1000 }, {"CM", 900}, {"D", 500}, {"CD", 400}, {"C", 100}, {"XC", 90},
{"L", 50}, {"XL", 40}, {"X", 10}, {"IX", 9}, {"V", 5}, {"IV", 4}, {"I", 1}
            };

            int result = 0;
            for(int i=0; i<s.Length;i++)
            {
                if (i != s.Length - 1 && romanMapping.ContainsKey(s[i] + "" + s[i + 1]))
                {
                    result = result + romanMapping[s[i] + "" + s[i + 1]];
                    i++;
                }
                else
                    result = result + romanMapping[s[i].ToString()];
            }
            return result;
    }
```

```
TC: O(n)
SC: O(1)
```

10. Letter Combinations of a Phone Number

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.



Example 1:

```
Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]
```

Example 2:

```
Input: digits = ""
Output: []
```

Example 3:

```
Input: digits = "2"
Output: ["a","b","c"]
```

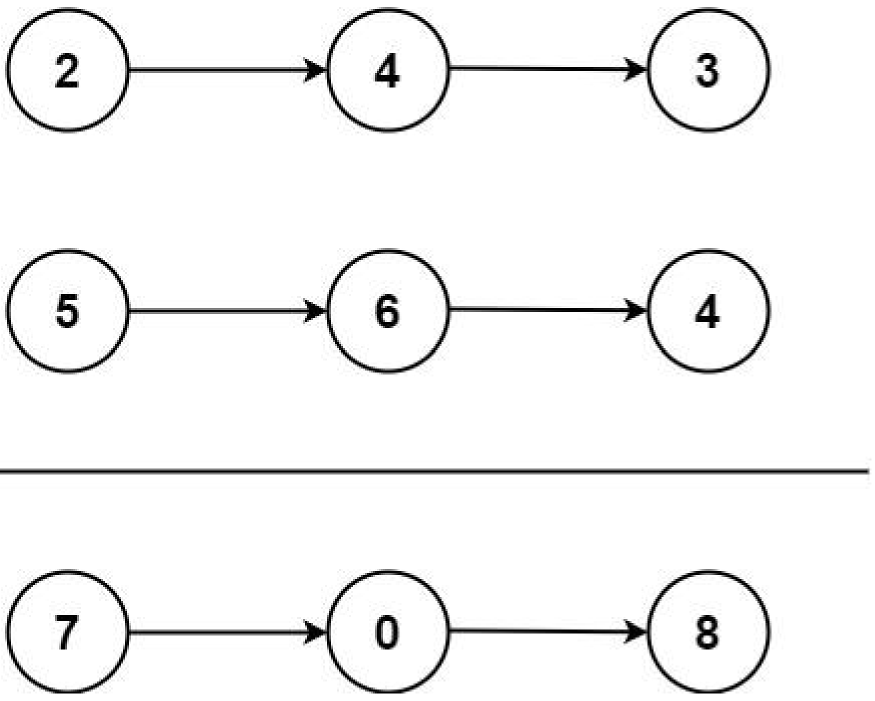
```
private List<string> ans = new List<string>();
        private string phoneDigits;
        private Dictionary<char, string> combos = new Dictionary<char, string> {
            { '2', "abc" }, {'3', "def" }, {'4', "ghi" }, {'5', "jkl" },
            { '6', "mno" }, {'7', "pqrs" }, {'8', "tuv" }, {'9', "wxyz" } };
        public IList<string> LetterCombinations(string digits)
        {
            if (digits.Length == 0)
                return ans;

            phoneDigits = digits;
            Helper(0, new StringBuilder());
            return ans;
        }

        private void Helper(int index, StringBuilder path)
        {
            if(path.Length == phoneDigits.Length)
            {
                ans.Add(path.ToString());
                return;
            }

            foreach(char c in combos[phoneDigits[index]])
            {
                path.Append(c);

                Helper(index + 1, path);

                path.Remove(path.Length - 1, 1);
            }
        }

TC: O(4^n n)
SC: O(1)
4 is the max length of the dictionary value (wxyz)
```

10. Add Two Numbers

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example 1:



```
Input: l1 = [2,4,3], l2 = [5,6,4]
Output: [7,0,8]
Explanation: 342 + 465 = 807.
```

Example 2:

```
Input: l1 = [0], l2 = [0]
Output: [0]
```

Example 3:

```
Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
Output: [8,9,9,9,0,0,0,1]
```

```
    public ListNode AddTwoNumbers(ListNode l1, ListNode l2) {
            ListNode ansListNode = new ListNode();
                ListNode curr = ansListNode;

                    int sum = 0;
                while (l1 != null || l2 != null)
                {
                    if(l1 != null)
                    {
                        sum += l1.val;
                        l1 = l1.next;
                    }
                    if (l2 != null)
                    {
                        sum += l2.val;
                        l2 = l2.next;
                    }
                    curr.next = new ListNode(sum % 10);
                    curr = curr.next;

                    sum = sum > 9 ? 1: 0;
                }

                if (sum != 0)
                    curr.next = new ListNode(sum);
                return ansListNode.next;
        }

    TC: O(max(m,n))
    SC: O(1)
```

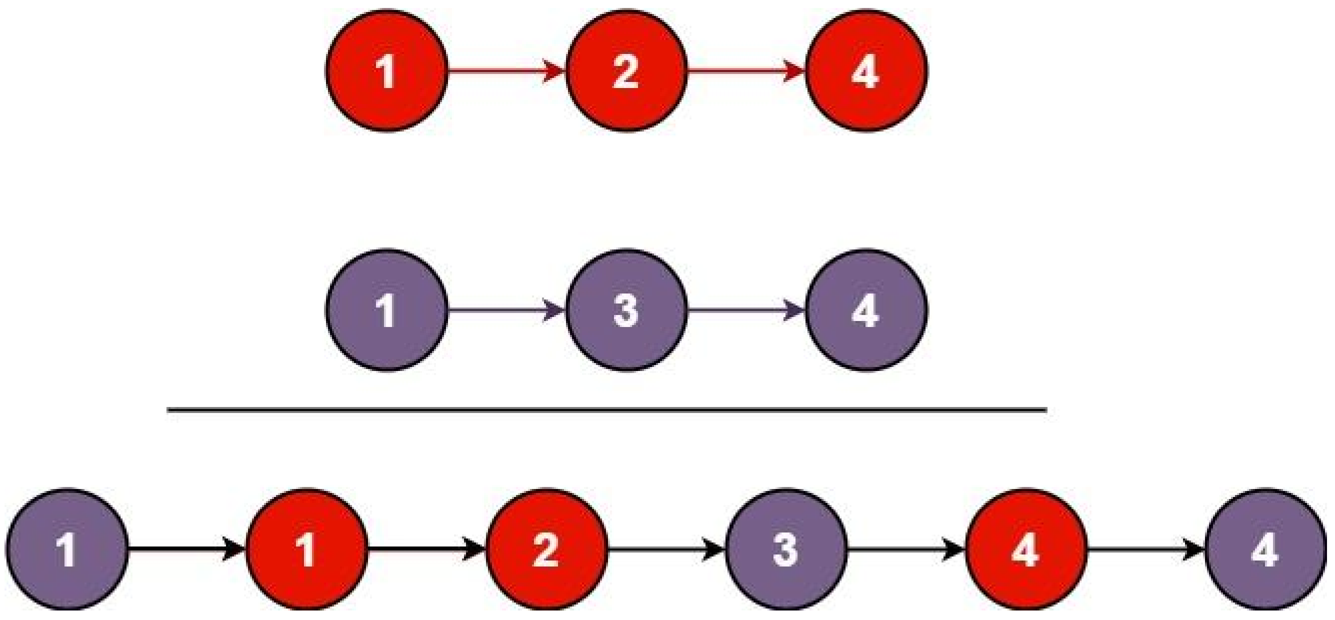
11. Merge Two Sorted Lists

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

Example 1:



```
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
```

Example 2:

```
Input: list1 = [], list2 = []
Output: []
```

Example 3:

```
Input: list1 = [], list2 = [0]
Output: [0]
```

```
public ListNode MergeTwoLists(ListNode list1, ListNode list2) {
        ListNode newNode = new ListNode(-1);
            ListNode curr = newNode ;
            while(list1 != null && list2 != null)
            {
                if(list1.val <= list2.val)
                {
                    curr.next = list1;
                    list1 = list1.next;
                }
                else
                {
                    curr.next = list2;
                    list2 = list2.next;
                }
                curr = curr.next;
            }

            curr.next = list1 != null ? list1 : list2;
            return newNode.next;
    }

TC: O(min(m,n))
SC: O(1)
```

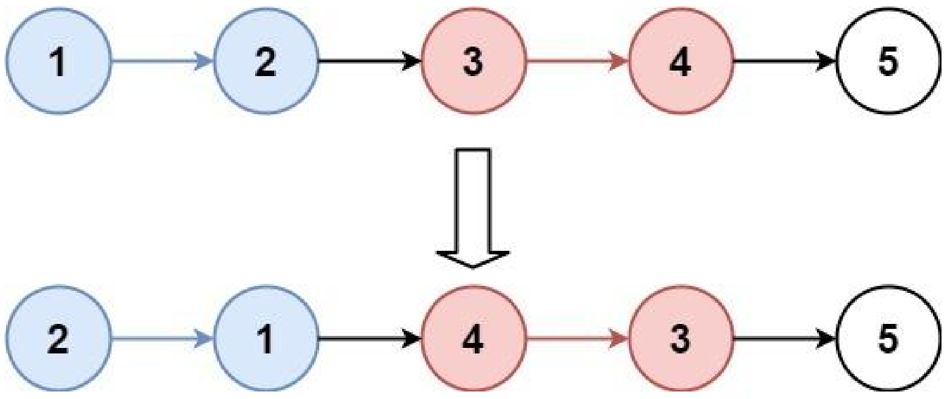
12. Reverse Nodes in k-Group

Given the head of a linked list, reverse the nodes of the list k at a time, and return *the modified list*.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should remain as it is.

You may not alter the values in the list's nodes, only nodes themselves may be changed.

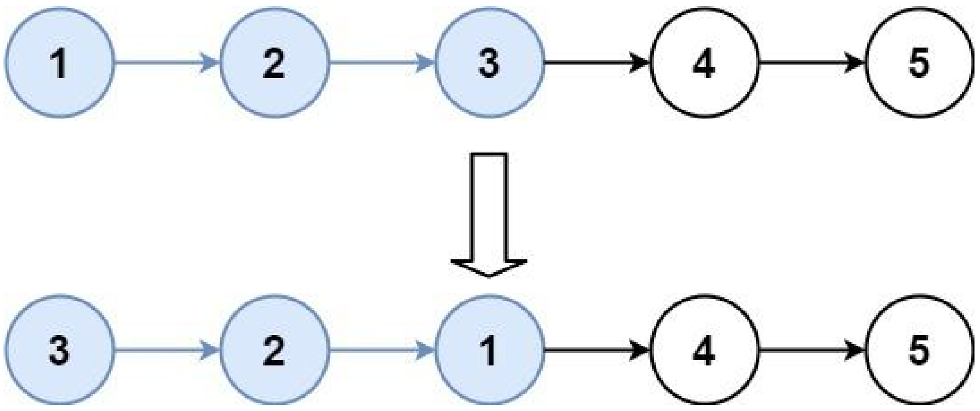
Example 1:



```
Input: head = [1,2,3,4,5], k = 2
```

```
Output: [2,1,4,3,5]
```

Example 2:



```
Input: head = [1,2,3,4,5], k = 3
Output: [3,2,1,4,5]
```

```
public ListNode ReverseKGroup(ListNode head, int k)
        {
            int count = 0;
            ListNode curr = head;

            // First, see if there are atleast k nodes
            // left in the linked list.
            while (count < k && curr != null)
            {
                curr = curr.next;
                count++;
            }

            if (count == k)
            {

                // Reverse the first k nodes of the list and
                // get the reversed list's head.
                ListNode reversedHead = this.ReverseLinkedList(head, k);

                // Now recurse on the remaining linked list. Since
                // our recursion returns the head of the overall processed
                // list, we use that and the "original" head of the "k" nodes
                // to re-wire the connections.
                head.next = this.ReverseKGroup(curr, k);
                return reversedHead;
            }

            return head;
        }
```

```
            private ListNode ReverseLinkedList(ListNode head, int k)
             {
                 ListNode prev = null;
                 ListNode next = prev;
                 ListNode curr = head;

                 while(k > 0)
                 {
                     next = curr.next;
                     curr.next = prev;
                     prev = curr;
                     curr = next;
                     k--;
                 }
                 return prev;
             }

     TC: O(n)
     SC: O(n/k)
```

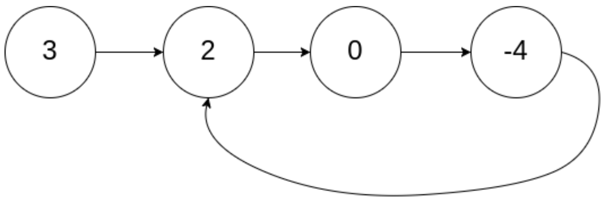
13. Linked List Cycle

Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter.

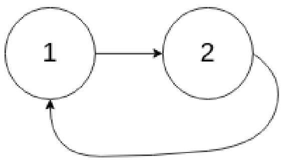
Return true *if there is a cycle in the linked list*. Otherwise, return false.

Example 1:



```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).
```

Example 2:



```
Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.
```

Example 3:

```
Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.
```

```
Solution 1: HashSet

public boolean hasCycle(ListNode head) {
        Set<ListNode> seen = new HashSet();

        while(head!=null)
        {
            if(seen.contains(head))
                return true;
            seen.add(head);
            head = head.next;
        }
        return false;
    }

TC: O(n)
SC: O(n)
```

```
Solution 2: Slow & Fast Pointer

public boolean hasCycle(ListNode head) {
        if(head == null)
            return false;

        ListNode slow = head;
        ListNode fast = head.next;

        while(slow!=fast)
        {
            if(fast == null || fast.next == null)
                return false;

            slow = slow.next;
            fast = fast.next.next;
        }
        return true;
    }

TC: O(n)
SC: O(1)
```

14. Generate Parentheses

Given `n` pairs of parentheses, write a function to *generate all combinations of well-formed parentheses*.

Example 1:

```
Input: n = 3
Output: ["((()))","(()())","(())()","()(())","()()()"]
```

Example 2:

```
Input: n = 1
Output: ["()"]
```

```
public List<String> GenerateParenthesis(int n)
        {
            List<string> ans = new List<string>();
            backtrack(ans, new StringBuilder(), 0, 0, n);
            return ans;
        }

        public void backtrack(List<String> ans, StringBuilder cur, int open, int close, int max)
        {
            if (cur.Length == max * 2)
            {
                ans.Add(cur.ToString());
                return;
            }

            if (open < max)
            {
                cur.Append("(");
                backtrack(ans, cur, open + 1, close, max);
                cur.Remove(cur.Length - 1, 1);
            }
            if (close < open)
            {
                cur.Append(")");
                backtrack(ans, cur, open, close + 1, max);
                cur.Remove(cur.Length - 1, 1);
            }
        }
```

TC: $O(4^n/\sqrt{n})$

SC: $O(4^n/\sqrt{n})$

15. Next Permutation

A permutation of an array of integers is an arrangement of its members into a sequence or linear order.

- For example, for arr = [1,2,3], the following are all the permutations of arr: [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1].

The next permutation of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the next permutation of that array is the permutation that follows it in the sorted container. If such arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

- For example, the next permutation of arr = [1,2,3] is [1,3,2].
- Similarly, the next permutation of arr = [2,3,1] is [3,1,2].
- While the next permutation of arr = [3,2,1] is [1,2,3] because [3,2,1] does not have a lexicographical larger rearrangement.

Given an array of integers nums, *find the next permutation of* nums.

The replacement must be in place and use only constant extra memory.

Example 1:

```
Input: nums = [1,2,3]
Output: [1,3,2]
```

Example 2:

```
Input: nums = [3,2,1]
Output: [1,2,3]
```

Example 3:

```
Input: nums = [1,1,5]
Output: [1,5,1]
```

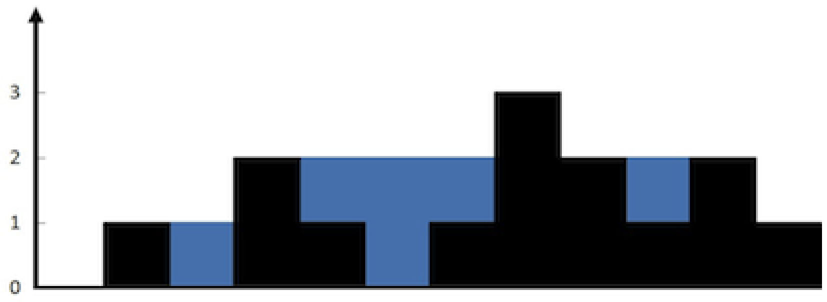
```
Hint: find the pattern
```

```
public void NextPermutation(int[] nums)
        {
            int i = nums.Length - 2;

            while (i >= 0 && nums[i + 1] <= nums[i])
                i--;

            if(i>=0)
            {
                int j = nums.Length - 1;
                while (nums[j] <= nums[i])
                    j--;

                Swap(i, j, nums);
            }
            Array.Reverse(nums, i + 1, nums.Length - i-1);
        }

        private void Swap(int i, int j, int []nums)
        {
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
        }

TC: O(n)
SC: O(1)
```

16. Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:



```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6
units of rain water (blue section) are being trapped.
```

Example 2:

```
Input: height = [4,2,0,3,2,5]
Output: 9
```

```
public int Trap(int[] height) {
        int left = 0, right = height.Length - 1;

            int leftMax = 0, rightMax = 0;
            int ans = 0;

            while(left<right)
            {
                if(height[left]<height[right])
                {
                    if (height[left] >= leftMax)
                        leftMax = height[left];
                    else
                        ans += (leftMax - height[left]);

                    left++;
                }
                else
                {
                    if (height[right] >= rightMax)
                        rightMax = height[right];
                    else
                        ans += (rightMax - height[right]);

                    right--;
                }
            }
            return ans;
```

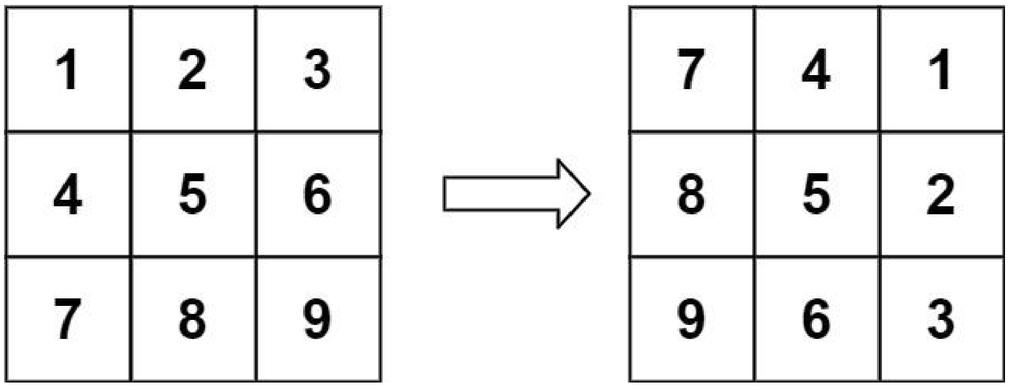
```
  }

TC: O(n)
SC: O(1)
```

17. Rotate Image

You are given an n x n 2D matrix representing an image, rotate the image by 90 degrees (clockwise).

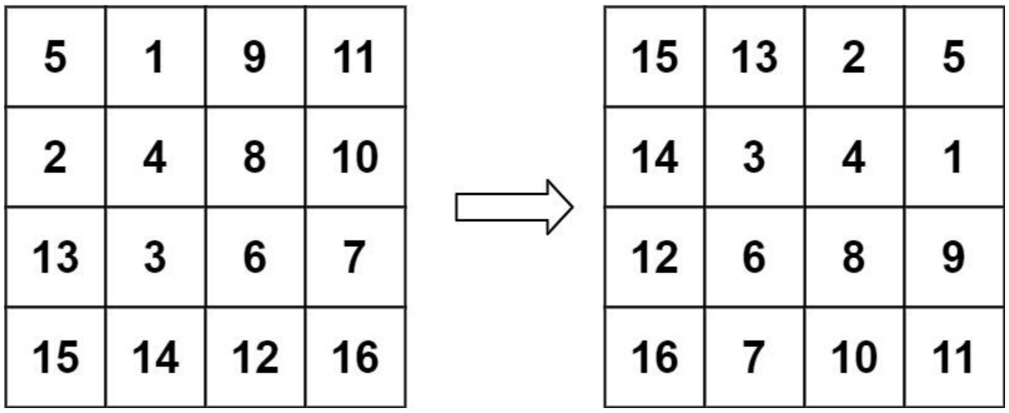
You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

Example 1:



```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]
```

Example 2:



```
Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]
```

```
public void Rotate(int[][] matrix) {
        int n = matrix.Length;
        for (int i = 0; i < (n + 1) / 2; i ++) {
            for (int j = 0; j < n / 2; j++) {
                int temp = matrix[n - 1 - j][i];
                matrix[n - 1 - j][i] = matrix[n - 1 - i][n - j - 1];
                matrix[n - 1 - i][n - j - 1] = matrix[j][n - 1 -i];
                matrix[j][n - 1 - i] = matrix[i][j];
                matrix[i][j] = temp;
            }
        }
    }

TC: O(m)
SC: O(1)
m -> number of cells
```

18. Group Anagrams

Given an array of strings strs, group the anagrams together. You can return the answer in any order.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

```
Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]
```

Example 2:

```
Input: strs = [""]
Output: [[""]]
```

Example 3:

```
Input: strs = ["a"]
Output: [["a"]]
```

```
public IList<IList<string>> GroupAnagrams(string[] strs)
     {
         Dictionary<string, List<string>> anagramCounter = new Dictionary<string, List<string>>();
         foreach (string s in strs)
         {
             char[] c = s.ToCharArray();
             Array.Sort(c);
             if (!anagramCounter.ContainsKey(new string(c)))
                 anagramCounter.Add(new string(c), new List<string> { s });
             else
                 anagramCounter[new string(c)].Add(s);
         }

         IList<IList<string>> result = new List<IList<string>>();
         foreach(List<string> group in anagramCounter.Values)
         {
             var list = new List<string>();
             list.AddRange(group);
             result.Add(list);
         }

         return result;
     }

  TC: O(n.k.logk)
  SC: O(n.k)
```

19. Find the Index of the First Occurrence in a String

Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Example 1:

```
Input: haystack = "sadbutsad", needle = "sad"
Output: 0
Explanation: "sad" occurs at index 0 and 6.
The first occurrence is at index 0, so we return 0.
```

Example 2:

```
Input: haystack = "leetcode", needle = "leeto"
Output: -1
Explanation: "leeto" did not occur in "leetcode", so we return -1.
```

```
public int StrStr(string haystack, string needle)
        {
            if (string.IsNullOrEmpty(needle))
                return 0;

            if (needle.Length > haystack.Length)
                return -1;
            int j = 0, index = -1,k=0 ;
            while (k<haystack.Length)
            {
                if (haystack[k] == needle[j])
                {
                    if (index == -1)
                        index = k;
                    j++;
                }
                else if (j > 0)
                {
                    j = 0;
                    k = index + 1;
                    index = -1;

                    continue;
                }
                k++;
                if (j == needle.Length)
                    return index;
            }

            return j != needle.Length ?-1: index;
        }
TC: O(n)
SC: O(1)
n -> length of haystack string
```

20. Maximum Subarray

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return *its sum*.

A subarray is a contiguous part of an array.

Example 1:

```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: [4,-1,2,1] has the largest sum = 6.
```

Example 2:

```
Input: nums = [1]
Output: 1
```

Example 3:

```
Input: nums = [5,4,-1,7,8]
Output: 23
```

```
public int MaxSubArray(int[] nums)
        {
            int currSubArray = nums[0];
            int maxSubArray = nums[0];

            for(int i=1;i<nums.Length;i++)
            {
                int num = nums[i];

                currSubArray = Math.Max(num, currSubArray + num);
                maxSubArray = Math.Max(maxSubArray, currSubArray);
            }

            return maxSubArray;
        }

TC: O(n)
SC: O(1)
```

21. Merge Intervals

Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input.*

Example 1:

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

Example 2:

```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

```
public int[][] Merge(int[][] intervals) {
        Comparer<int> comparer = Comparer<int>.Default;
            Array.Sort(intervals, (x, y) => comparer.Compare(x[0], y[0]));

            List<int[]> merge = new List<int[]>();

            foreach (int[] interval in intervals)
            {
                if (merge.Count == 0 || merge[merge.Count-1][1] < interval[0])
                {
                    merge.Add(interval);
                }
                else
                {
                    merge[merge.Count - 1][1] = Math.Max(merge[merge.Count - 1][1], interval[1]);
                }
            }
            return merge.ToArray();
        }

TC: O(nlogn)
SC: O(logn)
```

22. Simplify Path

Given a string path, which is an absolute path (starting with a slash '/') to a file or directory in a Unix-style file system, convert it to the simplified canonical path.

In a Unix-style file system, a period '.' refers to the current directory, a double period '..' refers to the directory up a level, and any multiple consecutive slashes (i.e. '//') are treated as a single slash '/'. For this problem, any other format of periods such as '...' are treated as file/directory names.

The canonical path should have the following format:

- The path starts with a single slash '/'.
- Any two directories are separated by a single slash '/'.
- The path does not end with a trailing '/'.
- The path only contains the directories on the path from the root directory to the target file or directory (i.e., no period '.' or double period '..')

Return *the simplified canonical path*.

Example 1:

```
Input: path = "/home/"
Output: "/home"
Explanation: Note that there is no trailing slash after the last directory name.
```

Example 2:

```
Input: path = "/../"
Output: "/"
Explanation: Going one level up from the root directory is a no-op, as the root level is the highest level
you can go.
```

Example 3:

```
Input: path = "/home//foo/"
Output: "/home/foo"
Explanation: In the canonical path, multiple consecutive slashes are replaced by a single one.
```

```
    public String simplifyPath(String path) {
            Stack<String> stack = new Stack<String>();
            String[] comp = path.split("/");

            for(String c : comp)
            {
                if(c.equals(".") || c.isEmpty())
                    continue;
                else if(c.equals(".."))
                {
                    if(!stack.isEmpty())
                        stack.pop();
                }
                else
                    stack.add(c);
            }
```

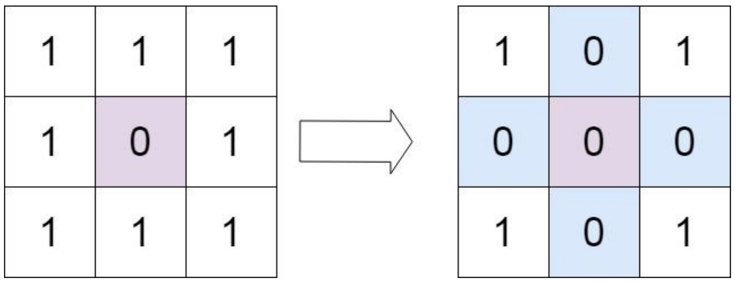
```
        StringBuilder sb = new StringBuilder();
        for(String s : stack)
        {
                sb.append('/');
                sb.append(s);
        }
        return sb.length() > 0 ? sb.toString() : "/" ;
    }

TC: O(n)
SC: (n)
```

23. Set Matrix Zeroes

Given an m x n integer matrix matrix, if an element is 0, set its entire row and column to 0's.

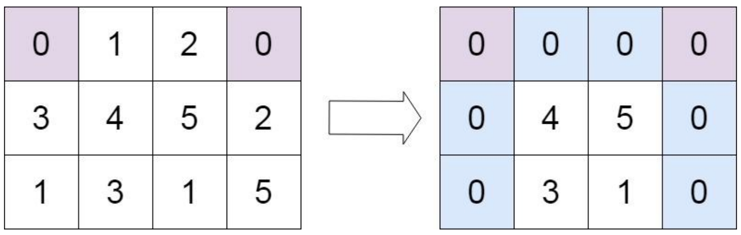
You must do it in place.

Example 1:



```
Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]
Output: [[1,0,1],[0,0,0],[1,0,1]]
```

Example 2:



```
Input: matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]
Output: [[0,0,0,0],[0,4,5,0],[0,3,1,0]]
```

```
Solution 1: Easy using Set
public void setZeroes(int[][] matrix) {
        int R = matrix.length;
        int C = matrix[0].length;

        Set<Integer> rows = new HashSet<>();
        Set<Integer> cols = new HashSet<>();

        for(int i=0; i<R; i++)
        {
            for(int j=0; j<C; j++)
            {
                if(matrix[i][j] == 0)
                {
                    rows.add(i);
                    cols.add(j);
                }
            }
        }

        for(int i=0; i<R; i++)
        {
            for(int j=0; j<C; j++)
            {
                if(rows.contains(i) || cols.contains(j))
                    matrix[i][j] = 0;
            }
        }
    }
```

```
TC: O(m*n)
SC: O(m+n)
m -> number of rows
n -> number of columns
```

Solution 2: O(1) Space Complexity

```
public void SetZeroes(int[][] matrix) {
        bool isCol = false;
            int R = matrix.Length;
            int C = matrix[0].Length;

            for (int i = 0; i < R; i++)
            {
                if (matrix[i][0] == 0)
                {
                    isCol = true;
                }

                for (int j = 1; j < C; j++)
                {
                    if (matrix[i][j] == 0)
                    {
                        matrix[0][j] = 0;
                        matrix[i][0] = 0;
                    }
                }
            }
            for (int i = 1; i < R; i++)
            {
                for (int j = 1; j < C; j++)
                {
                    if (matrix[i][0] == 0 || matrix[0][j] == 0)
                    {
                        matrix[i][j] = 0;
                    }
                }
            }

            if (matrix[0][0] == 0)
            {
                for (int j = 0; j < C; j++)
                {
                    matrix[0][j] = 0;
                }
            }

            if (isCol)
            {
                for (int i = 0; i < R; i++)
                {
                    matrix[i][0] = 0;
                }
            }
        }

TC: O(m*n)
SC: O(1)
```

24. Minimum Window Substring

Given two strings `s` and `t` of lengths `m` and `n` respectively, return *the minimum window substring of* `s` *such that every character in* `t` *(including duplicates) is included in the window. If there is no such substring, return the empty string* `""`.

The testcases will be generated such that the answer is unique.

A substring is a contiguous sequence of characters within the string.

Example 1:

```
Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.
```

Example 2:

```
Input: s = "a", t = "a"
Output: "a"
Explanation: The entire string s is the minimum window.
```

Example 3:

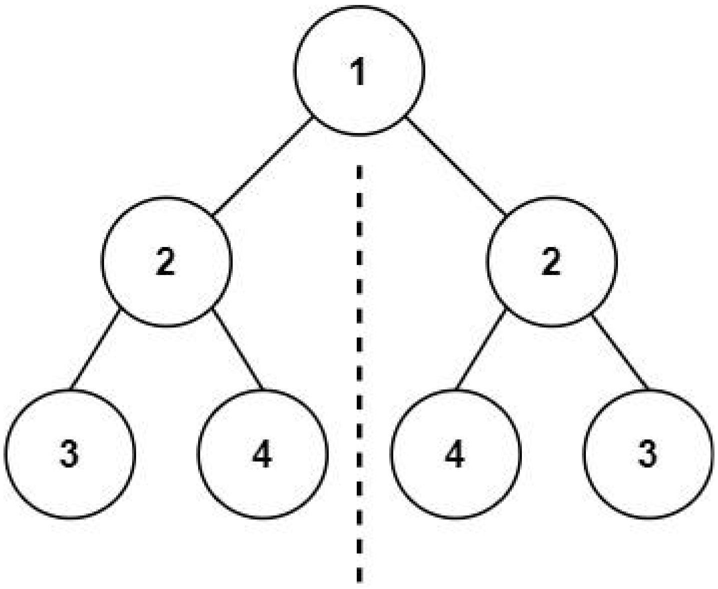
```
Input: s = "a", t = "aa"
Output: ""
Explanation: Both 'a's from t must be included in the window.
Since the largest window of s only has one 'a', return empty string.
```

```
public string MinWindow(string s, string t)
        {
            if (s.Length == 0 || t.Length == 0)
                return "";
            Dictionary<char, int> dictT = new Dictionary<char, int>();
            foreach(char c in t)
            {
                if (dictT.TryGetValue(c, out int value))
                    dictT[c] = value+1;
                else
                    dictT.Add(c, 1);
            }

            int required = dictT.Count;
            int l = 0;
            int r = 0;

            Dictionary<char, int> windowCount = new Dictionary<char, int>();
            int count = 0;

            int[] ans = new int[] { -1, 0, 0 };

            while(r<s.Length)
            {
                char c = s[r];
                if (windowCount.TryGetValue(c, out int value))
                {
                    windowCount[c] = value + 1;
                }
                else
                    windowCount.Add(c, 1);

                if (dictT.ContainsKey(c) && windowCount[c] == dictT[c])
                    count++;

                while(l<=r && count == required)
                {
                    c = s[l];

                    if(ans[0] == -1 || ans[0] > r-l+1)
                    {
                        ans[0] = r - l + 1;
                        ans[1] = l;
                        ans[2] = r;
                    }

                    windowCount[c] = windowCount[c] - 1;
                    if (dictT.ContainsKey(c) && windowCount[c] < dictT[c])
                        count--;

                    l++;
                }

                r++;

            }

            return ans[0] == -1 ? "" : s.Substring(ans[1], ans[2] - ans[1]  + 1);
        }

TC: O(|S| + |T|)
SC: O(|S| + |T|)
```

25. Symmetric Tree

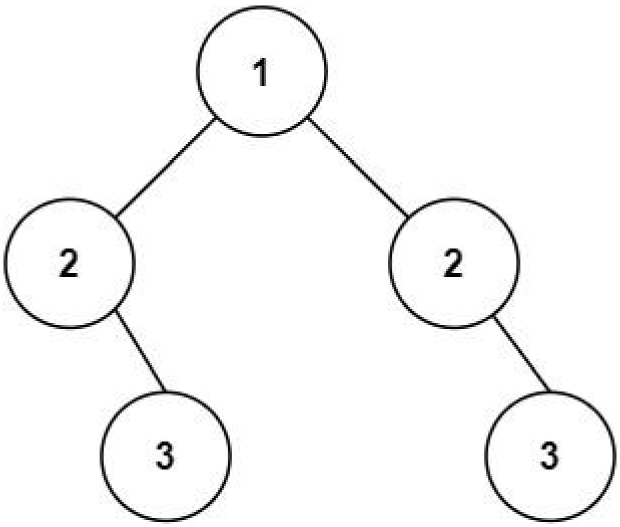
Given the root of a binary tree, *check whether it is a mirror of itself* (i.e., symmetric around its center).

Example 1:



```
Input: root = [1,2,2,3,4,4,3]
Output: true
```

Example 2:



```
Input: root = [1,2,2,null,3,null,3]
```

```
Output: false
```

Solution 1: Using Recursion

```
public boolean isSymmetric(TreeNode root) {
        return isMirror(root, root);
    }

    private boolean isMirror(TreeNode t1, TreeNode t2){
        if(t1 == null && t2 == null)
            return true;

        if(t1 == null || t2 == null)
            return false;

        return (t1.val == t2.val) && isMirror(t1.right, t2.left) && isMirror(t1.left, t2.right);
    }
```

TC: O(n)
SC: O(n)

Solution 2: Using Loop and Queue

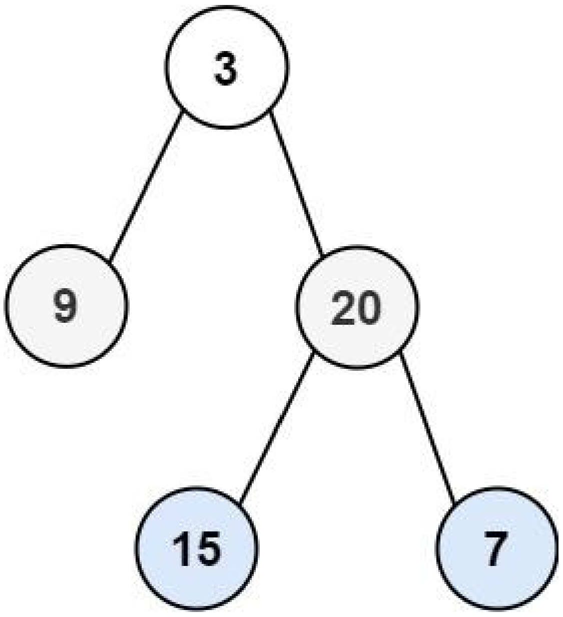
```
public bool IsSymmetric(TreeNode root)
        {
            Queue<TreeNode> treeQueue = new Queue<TreeNode>();
            treeQueue.Enqueue(root);
            treeQueue.Enqueue(root);

            while(treeQueue.Count != 0)
            {
                TreeNode tree1 = treeQueue.Dequeue();
                TreeNode tree2 = treeQueue.Dequeue();

                if (tree1 == null && tree2 == null)
                    continue;
                if (tree1 == null || tree2 == null)
                    return false;
                if (tree1.val != tree2.val)
                    return false;

                treeQueue.Enqueue(tree1.left);
                treeQueue.Enqueue(tree2.right);
                treeQueue.Enqueue(tree1.right);
                treeQueue.Enqueue(tree2.left);
            }
            return true;
        }
```

TC: O(n)
SC: O(n)

26. Binary Tree Level Order Traversal

Given the root of a binary tree, return *the level order traversal of its nodes' values*. (i.e., from left to right, level by level).

Example 1:



```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[9,20],[15,7]]
```

Example 2:

```
Input: root = [1]
Output: [[1]]
```

Example 3:

```
Input: root = []
Output: []
```

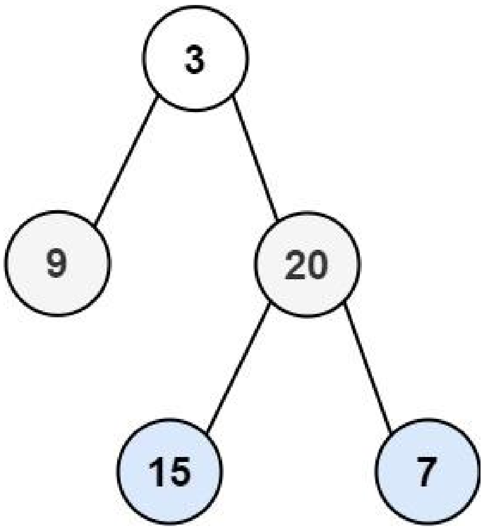
```
Solution 1: Using Recursion

    IList<IList<int>> levels = new List<IList<int>>();

    public IList<IList<int>> LevelOrder(TreeNode root)
        {
            if(root == null)
                return levels;
            Helper(root, 0);
            return levels;
        }

    private void Helper(TreeNode node, int level)
    {
        if(levels.Count == level)
            levels.Add(new List<int>());

        levels[level].Add(node.val);

        if(node.left != null)
```

```
                Helper(node.left, level+1);
            if(node.right != null)
                Helper(node.right, level+1);
        }

TC: O(n)
SC: O(n)



Solution 2: Using Queue & Loop

public IList<IList<int>> LevelOrder(TreeNode root)
        {
            Queue<TreeNode> q = new Queue<TreeNode>();
            IList<IList<int>> ans = new List<IList<int>>();
            if (root == null)
                return ans;

            q.Enqueue(root);
            int level = 0;
            while(q.Count != 0)
            {
                ans.Add(new List<int>());

                int size = q.Count;
                for(int i=0; i<size; i++)
                {
                    TreeNode val = q.Dequeue();
                    ans[level].Add(val.val);
                    if (val.left != null)
                        q.Enqueue(val.left);
                    if (val.right != null)
                        q.Enqueue(val.right);
                }
                level++;
            }
            return ans;
        }
TC: O(n)
SC: O(n)
```

27. Binary Tree Zigzag Level Order Traversal

Given the root of a binary tree, return *the zigzag level order traversal of its nodes' values*. (i.e., from left to right, then right to left for the next level and alternate between).

Example 1:



```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[20,9],[15,7]]
```

Example 2:

```
Input: root = [1]
Output: [[1]]
```

Example 3:

```
Input: root = []
Output: []
```

```
public class Solution {
    IList<IList<int>> ans = new List<IList<int>>();
    public IList<IList<int>> ZigzagLevelOrder(TreeNode root) {
       if (root == null)
                return ans;
            helper(root, 0);
            return ans;
        }

        private void helper(TreeNode root, int level)
        {
            if (ans.Count == level)
                ans.Add(new List<int>());

            ans[level].Add(root.val);

            if (level % 2 != 0)
            {
                if (root.left != null)
                    helper(root.left, level + 1);
                if (root.right != null)
                    helper(root.right, level + 1);
            }
            else
            {
                if (root.right != null)
                    helper(root.right, level + 1);
                if (root.left != null)
                    helper(root.left, level + 1);
            }
        }
}
```

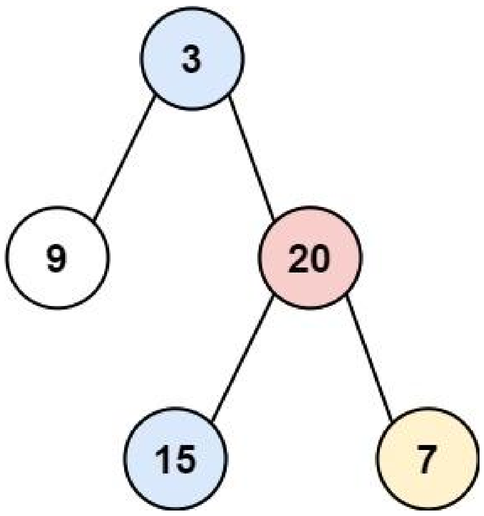
```
TC: O(n)
SC: O(n)
```

28. Binary Tree Vertical Order Traversal

Given the root of a binary tree, return *the vertical order traversal of its nodes' values*. (i.e., from top to bottom, column by column).

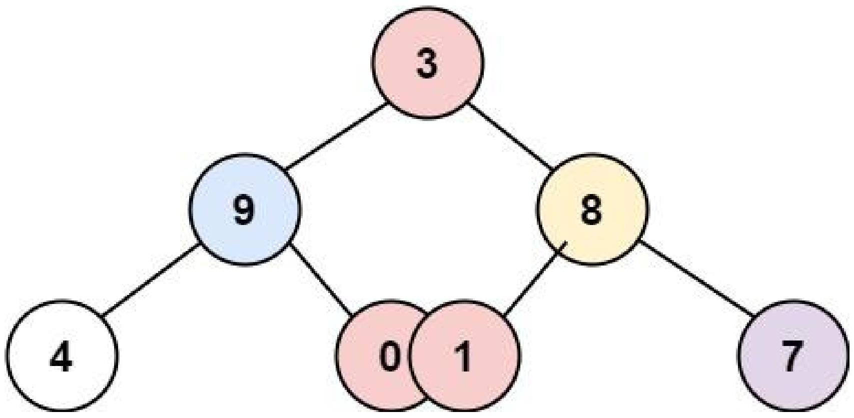
If two nodes are in the same row and column, the order should be from left to right.

Example 1:



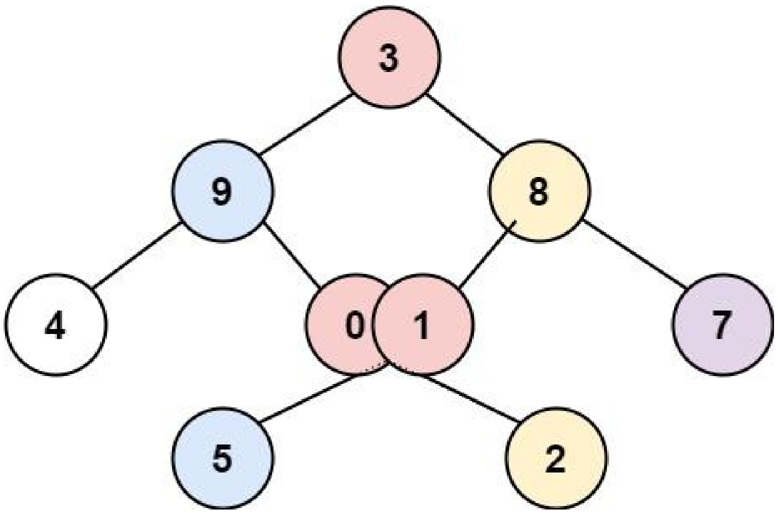
```
Input: root = [3,9,20,null,null,15,7]
Output: [[9],[3,15],[20],[7]]
```

Example 2:



```
Input: root = [3,9,8,4,0,1,7]
Output: [[4],[9],[3,0,1],[8],[7]]
```

Example 3:



```
Input: root = [3,9,8,4,0,1,7,null,null,null,2,5]
Output: [[4],[9,5],[3,0,1],[8,2],[7]]
```

```
public List<List<Integer>> verticalOrder(TreeNode root) {

        List<List<Integer>> output = new ArrayList();

        if(root == null)
            return output;

        Map<Integer, ArrayList> columnTable = new HashMap();
        Queue<Pair<TreeNode, Integer>> q = new ArrayDeque();

        int column = 0;
        int minColumn = 0;
        int maxColumn = 0;
        q.offer(new Pair(root, column));

        while(!q.isEmpty())
        {
            Pair<TreeNode, Integer> p = q.poll();
            root = p.getKey();
            column = p.getValue();

            if(root != null)
            {
                if(!columnTable.containsKey(column))
                    columnTable.put(column, new ArrayList());

                columnTable.get(column).add(root.val);

                minColumn = Math.min(minColumn, column);
                maxColumn = Math.max(maxColumn, column);

                q.offer(new Pair(root.left, column-1));
                q.offer(new Pair(root.right, column+1));
            }
        }

        for(int i=minColumn; i<maxColumn+1; i++)
        {
            output.add(columnTable.get(i));
        }
        return output;
    }

TC: O(n)
SC: O(n)
```

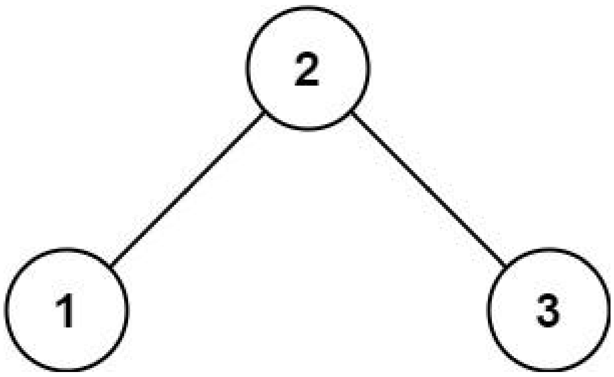
29. Validate Binary Search Tree

Given the root of a binary tree, *determine if it is a valid binary search tree (BST)*.

A valid BST is defined as follows:

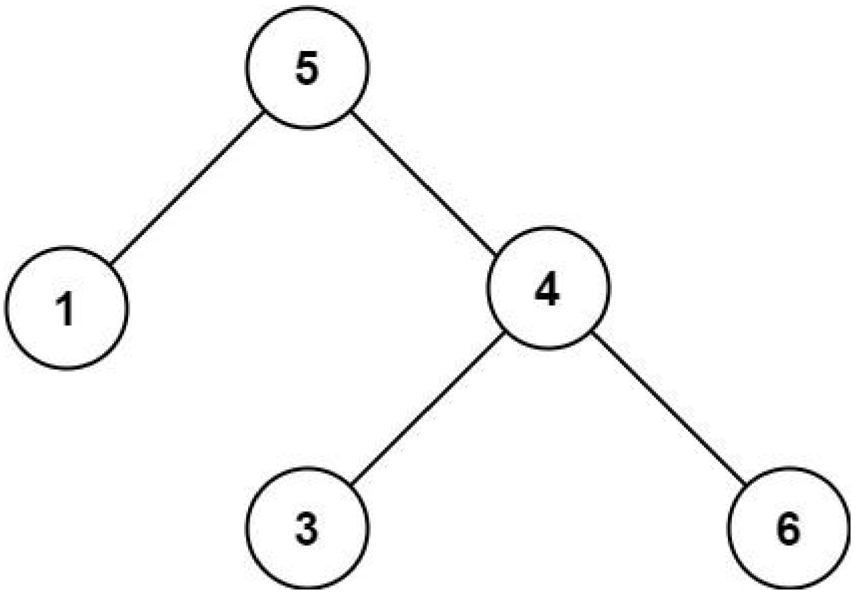
- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:



```
Input: root = [2,1,3]
Output: true
```

Example 2:



```
Input: root = [5,1,4,null,null,3,6]
Output: false
Explanation: The root node's value is 5 but its right child's value is 4.
```

```
    public bool IsValidBST(TreeNode root)
            {
```

```
            return Validate(root, null, null);
        }

        public bool Validate(TreeNode root, int? low, int? high)
        {
            if (root == null)
                return true;

            if (low != null && root.val <= low || high != null && root.val >= high)
                return false;

            return (Validate(root.left, low, root.val) && Validate(root.right, root.val, high));
        }

TC: O(n)
SC: O(n)
```

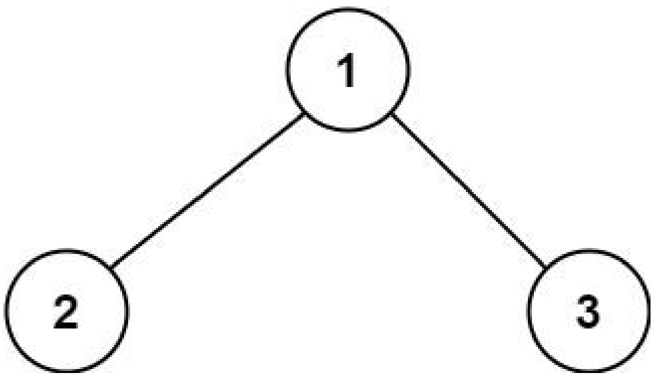
30. Binary Tree Maximum Path Sum

A path in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence at most once. Note that the path does not need to pass through the root.

The path sum of a path is the sum of the node's values in the path.

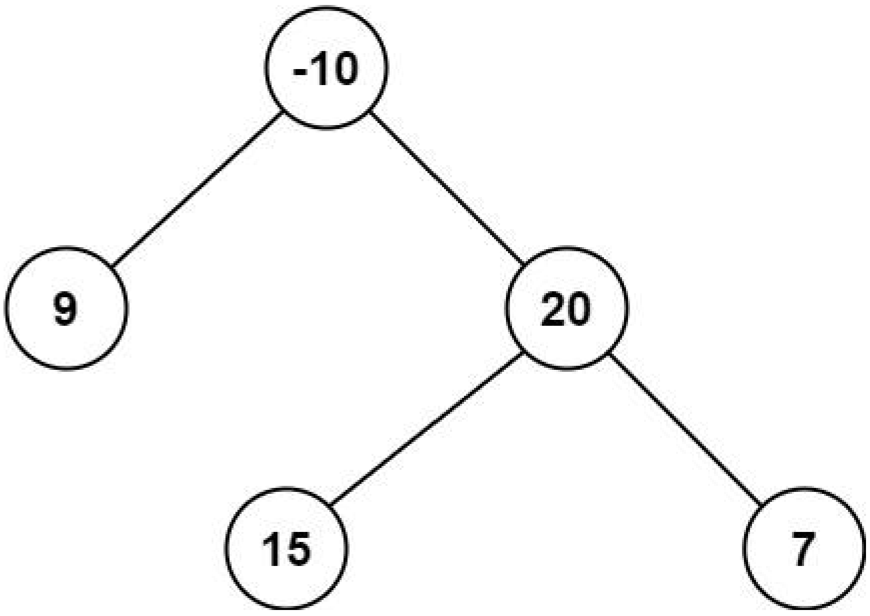
Given the root of a binary tree, return *the maximum path sum of any non-empty path*.

Example 1:



```
Input: root = [1,2,3]
Output: 6
Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.
```

Example 2:



```
Input: root = [-10,9,20,null,null,15,7]
Output: 42
Explanation: The optimal path is 15 -> 20 -> 7 with a path sum of 15 + 20 + 7 = 42.
```

```
public class Solution {
    public int max = int.MinValue;
        public int MaxPathSum(TreeNode root)
        {
            Helper(root);

            return max;
        }

        private int Helper(TreeNode root)
        {
            if (root == null)
                return 0;

            int leftGain = Math.Max(Helper(root.left), 0);
            int rightGain = Math.Max(Helper(root.right), 0);

            int newPath = root.val + leftGain + rightGain;

            max = Math.Max(max, newPath);

            return root.val + Math.Max(leftGain, rightGain);
        }
}

TC: O(n)
SC: o(H)

H -> height of the tree
Average case (balanced tree) H = log n
```

```
Worst case (Skewed tree) H = n
```

31. Word Ladder

A transformation sequence from word beginWord to word endWord using a dictionary wordList is a sequence of words beginWord -> s1 -> s2 -> ... -> sk such that:

- Every adjacent pair of words differs by a single letter.
- Every si for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordList.
- sk == endWord

Given two words, beginWord and endWord, and a dictionary wordList, return *the number of words in the shortest transformation sequence from* beginWord *to* endWord*, or* 0 *if no such sequence exists.*

Example 1:

```
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> cog", which is 5
words long.
```

Example 2:

```
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]
Output: 0
Explanation: The endWord "cog" is not in wordList, therefore there is no valid transformation sequence.
```

```
public int LadderLength(string beginWord, string endWord, IList<string> wordList)
        {
            Queue<string> queue = new Queue<string>();
            HashSet<string> words = new HashSet<string>(wordList);

            queue.Enqueue(beginWord);
            words.Remove(beginWord);

            int level = 0;
            while(queue.Count != 0)
            {
                int size = queue.Count;
                level++;

                for (int i = 0; i < size; i++)
                {
                    string word = queue.Dequeue();
                    if (word.Equals(endWord))
                        return level;
                    List<string> neighbors = Neighbors(word);
                    foreach (string neigh in neighbors)
                    {
                        if (words.Contains(neigh))
                        {
                            queue.Enqueue(neigh);
                            words.Remove(neigh);
                        }
                    }
                }
            }
            return 0;
        }

        private List<string> Neighbors(string word)
        {
            char[] w = word.ToCharArray();
            List<string> neigh = new List<string>();
            for (int i = 0; i < word.Length; i++)
            {
                char temp = w[i];
                for (char c = 'a'; c <= 'z'; c++)
                {
                    w[i] = c;
                    neigh.Add(new String(w));
                }
                w[i] = temp;
            }
            return neigh;
        }
```

TC: $O(M^2 * N)$

SC: $O(M^2 * N)$

32. Best Time to Buy and Sell Stock

You are given an array prices where prices[i] is the price of a given stock on the ith day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return 0.

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
```

Example 2:

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.
```

```
public int MaxProfit(int[] prices)
        {
            int minPrice = int.MaxValue;
            int maxProfit = 0;

            foreach(int price in prices)
            {
                if (price < minPrice)
                    minPrice = price;
                else if (price - minPrice > maxProfit)
                    maxProfit = price - minPrice;
            }
            return maxProfit;
        }

TC: O(n)
SC: O(1)
```

33. 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 the reversed string.
```

```
public string ReverseWords(string s) {
        string[] words = s.Trim().Split(" ").Where(x => x != "").ToArray();

            int left = 0;
            int right = words.Length - 1;

            string[] ansWords = new string[words.Length];
            while(left<right)
            {
                ansWords[left] = words[right];
                ansWords[right] = words[left];
                left++;
                right--;
            }

            int mid = words.Length / 2;
            if (words.Length % 2 != 0)
                ansWords[mid] = words[mid];
            return string.Join(" ", ansWords);
    }

TC: O(n)
SC: O(1)
```

34. Compare Version Numbers

Given two version numbers, version1 and version2, compare them.

Version numbers consist of one or more revisions joined by a dot '.'. Each revision consists of digits and may contain leading zeros. Every revision contains at least one character. Revisions are 0-indexed from left to right, with the leftmost revision being revision 0, the next revision being revision 1, and so on. For example 2.5.33 and 0.1 are valid version numbers.

To compare version numbers, compare their revisions in left-to-right order. Revisions are compared using their integer value ignoring any leading zeros. This means that revisions 1 and 001 are considered equal. If a version number does not specify a revision at an index, then treat the revision as 0. For example, version 1.0 is less than version 1.1 because their revision 0s are the same, but their revision 1s are 0 and 1 respectively, and 0 < 1.

Return the following:

- If version1 < version2, return -1.
- If version1 > version2, return 1.
- Otherwise, return 0.

Example 1:
```
Input: version1 = "1.01", version2 = "1.001"
Output: 0
Explanation: Ignoring leading zeroes, both "01" and "001" represent the same integer "1".
```

Example 2:
```
Input: version1 = "1.0", version2 = "1.0.0"
Output: 0
Explanation: version1 does not specify revision 2, which means it is treated as "0".
```

Example 3:
```
Input: version1 = "0.1", version2 = "1.1"
Output: -1
Explanation: version1's revision 0 is "0", while version2's revision 0 is "1". 0 < 1, so version1 < version2.
```

```
public int CompareVersion(string version1, string version2) {
        List<string> version1Split = version1.Split(".").ToList();
            List<string> version2Split = version2.Split(".").ToList();

            for(int i=0; i<version1Split.Count && version1Split.Count > version2Split.Count; i++)
            {
                version2Split.Add("0");
            }
            for (int i = 0; i < version1Split.Count && version2Split.Count > version1Split.Count;
i++)
            {
                version1Split.Add("0");
            }

            for (int i=0;i<version1Split.Count;i++)
            {
                int v1 = int.Parse(version1Split[i]);
                int v2 = int.Parse(version2Split[i]);

                if (v1 < v2)
                    return -1;
                else if (v1 > v2)
                    return 1;
            }
            return 0;
```

```
    }

TC: O(max(m,n))
SC: O(m+n)
```

35. Reverse Words in a String II

Given a character array 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 a single space.

Your code must solve the problem in-place, i.e. without allocating extra space.

Example 1:

```
Input: s = ["t","h","e"," ","s","k","y"," ","i","s"," ","b","l","u","e"]
Output: ["b","l","u","e"," ","i","s"," ","s","k","y"," ","t","h","e"]
```

Example 2:

```
Input: s = ["a"]
Output: ["a"]
```

```
public void ReverseWords(char[] s) {
        new Solution().ReverseString(s, 0, s.Length-1);

            int start = 0, end = 0;

            while(start<s.Length)
            {
                while(end<s.Length && s[end] != ' ')
                {
                    end++;
                }
                new Solution().ReverseString(s, start, end-1);
                start = end + 1;
                end++;
            }
    }
    public void ReverseString(char[] s, int left, int right)
        {
            while (left < right)
            {
                char temp = s[left];
                s[left] = s[right];
                left++;
                s[right] = temp;
                right--;
                }
            }
        TC: O(n)
        SC: O(1)
```

36. Number of Islands

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return *the number of islands*.

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
Input: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
Output: 1
```

Example 2:

```
Input: grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
Output: 3
```

```
        public int NumIslands(char[][] grid)
            {
                if (grid == null || grid.Length == 0)
                    return 0;

                int nr = grid.Length;
                int nc = grid[0].Length;
                int numOfIslands = 0;
                for(int i=0; i<nr;i++)
                {
                    for(int j=0;j<nc;j++)
                    {
                        if(grid[i][j] == '1')
                            numOfIslands++;
                        DFS(grid, i, j);
                    }
                }
                return numOfIslands;
            }

            private void DFS(char[][] grid, int r, int c)
            {
                int nr = grid.Length;
                int nc = grid[0].Length;
                if (r < 0 || c < 0 || r >= nr || c >= nc || grid[r][c] == '0')
                    return;

                grid[r][c] = '0';
                DFS(grid, r + 1, c);
                DFS(grid, r - 1, c);
                DFS(grid, r, c + 1);
                DFS(grid, r, c - 1);
            }

    TC: O(m*n)
    SC: O(m*n)
```

37. Course Schedule

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course bi first if you want to take course ai.

- For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.

Return true if you can finish all courses. Otherwise, return false.

Example 1:

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0. So it is possible.
```

Example 2:

```
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0, and to take course 0 you should also have finished course
1. So it is impossible.
```

```
public bool CanFinish(int numCourses, int[][] prerequisites)
        {
            Dictionary<int, List<int>> adjList = new Dictionary<int, List<int>>();

            foreach(int []i in prerequisites)
            {
                if(adjList.TryGetValue(i[1], out List<int> dependents))
                {
                    dependents.Add(i[0]);
                }
                else
                {
                    adjList.Add(i[1], new List<int> { i[0] });
                }
            }

            bool []paths = new bool[numCourses];
            bool []check = new bool[numCourses];

            for(int i=0;i<numCourses;i++)
            {
                if (IsCyclic(i, adjList, paths, check))
```

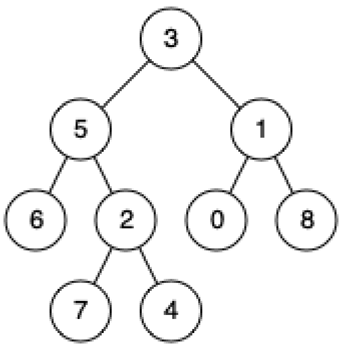
```
                    return false;
            }
            return true;
        }

        private bool IsCyclic(int currCourse, Dictionary<int, List<int>> adjList, bool []paths,
bool []check)
        {
            if (check[currCourse])
                return false;

            if (paths[currCourse])
                return true;

            if (!adjList.ContainsKey(currCourse))
                return false;

            paths[currCourse] = true;

            bool ret = false;
            foreach(int i in adjList[currCourse])
            {
                ret = IsCyclic(i, adjList, paths, check);
                if (ret)
                    break;
            }

            check[currCourse] = true;
            paths[currCourse] = false;
            return ret;
        }

TC: O(|E|+|V|)
SC: O(|E|+|V|)
V -> Number of Courses
E -> Number of dependencies
```

38. Lowest Common Ancestor of a Binary Tree

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

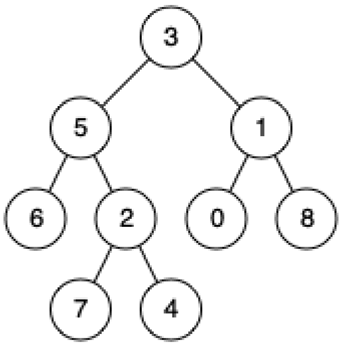
According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself).”

Example 1:



```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.
```

Example 2:



```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
Output: 5
Explanation: The LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the LCA definition.
```

Example 3:

```
Input: root = [1,2], p = 1, q = 2
Output: 1
```

```
 Solution 1: Using Recursion
TreeNode answer;
public TreeNode LowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    RecurseTree(root, p, q);
    return answer;
}

private bool RecurseTree(TreeNode curr, TreeNode p, TreeNode q){
    if(curr == null)
        return false;

    int left = RecurseTree(curr.left, p, q) ? 1: 0;
```

```
            int right = RecurseTree(curr.right, p, q) ? 1: 0;

            int mid = (curr == p || curr == q) ? 1: 0;

            if(mid + left + right >= 2)
                answer = curr;
            return (mid + left + right > 0);
        }

TC: O(n)
SC: O(n)


Solution 2: Using Stack & Loop
public TreeNode LowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q)
        {
            Stack<TreeNode> stack = new Stack<TreeNode>();
            Dictionary<TreeNode, TreeNode> parents = new Dictionary<TreeNode, TreeNode>();
            stack.Push(root);
            parents.Add(root, null);
            while(!parents.ContainsKey(p) || !parents.ContainsKey(q))
            {
                TreeNode node = stack.Pop();
                if(node.left != null)
                {
                    parents.Add(node.left, node);
                    stack.Push(node.left);
                }
                if(node.right != null)
                {
                    parents.Add(node.right, node);
                    stack.Push(node.right);
                }
            }
            HashSet<TreeNode> set = new HashSet<TreeNode>();
            while(p != null)
            {
                set.Add(p);
                p = parents[p];
            }

            while(!set.Contains(q))
            {
                q = parents[q];
            }

            return q;
        }

TC: O(n)
SC: O(n)
```

39. Product of Array Except Self

Given an integer array `nums`, return *an array* `answer` *such that* `answer[i]` *is equal to the product of all the elements of* `nums` *except* `nums[i]`.

The product of any prefix or suffix of `nums` is guaranteed to fit in a 32-bit integer.

You must write an algorithm that runs in `O(n)` time and without using the division operation.

Example 1:

```
Input: nums = [1,2,3,4]
Output: [24,12,8,6]
```

Example 2:

```
Input: nums = [-1,1,0,-3,3]
Output: [0,0,9,0,0]
```

```
public int[] ProductExceptSelf(int[] nums) {

            int length = nums.Length;
            int[] answer = new int[length];

            answer[0] = 1;
            for (int i = 1; i < length; i++)
            {
                answer[i] = answer[i - 1] * nums[i - 1];
            }

            int right = 1;
            for (int i = length - 1; i >= 0; i--)
            {
                answer[i] = answer[i] * right;
                right = right*nums[i];
            }

            return answer;
        }

TC: O(n)
SC: O(1)
```

40. Coin Change

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.

Return *the fewest number of coins that you need to make up that amount*. If that amount of money cannot be made up by any combination of the coins, return -1.

You may assume that you have an infinite number of each kind of coin.

Example 1:

```
Input: coins = [1,2,5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1
```

Example 2:

```
Input: coins = [2], amount = 3
Output: -1
```

Example 3:

```
Input: coins = [1], amount = 0
Output: 0
```

```
public int CoinChange(int[] coins, int amount) {
        int[] dp = new int[amount + 1];
            Array.Fill(dp, amount + 1);

            dp[0] = 0;
            for(int i=1; i<=amount; i++)
            {
                foreach(int coin in coins)
                {
                    if (i - coin < 0)
                        continue;

                    dp[i] = Math.Min(dp[i], dp[i - coin] + 1);
                }
            }
            return dp[amount] == (amount + 1) ? -1 : dp[amount];
    }

TC: O(s*n)
SC: O(s)
s -> amount
n -> number of denominations
```

41. Reconstruct Original Digits from English

Given a string `s` containing an out-of-order English representation of digits `0-9`, return *the digits in ascending order*.

Example 1:

```
Input: s = "owoztneoer"
Output: "012"
```

Example 2:

```
Input: s = "fviefuro"
Output: "45"
```

```
public String originalDigits(String s) {
        char[] count = new char[26+(int)'a'];

        for(char c : s.toCharArray())
        {
            count[c]++;
        }

        int[] out = new int[10];
        out[0] = count['z'];
        out[2] = count['w'];
        out[4] = count['u'];
        out[6] = count['x'];
        out[8] = count['g'];

        out[3] = count['h'] - out[8];
        out[5] = count['f'] - out[4];
        out[7] = count['s'] - out[6];

        out[9] = count['i'] - out[5] - out[6] - out[8];
        out[1] = count['n'] - out[7] - 2*out[9];

        StringBuilder sb = new StringBuilder();

        for(int i=0; i<10; i++)
        {
            for(int j=0; j<out[i]; j++)
            {
                sb.append(i);
            }
        }
        return sb.toString();
    }

TC: O(n)
SC: O(1)
```

42. Subarray Sum Equals K

Given an array of integers nums and an integer k, return *the total number of subarrays whose sum equals to* k.

Example 1:

```
Input: nums = [1,1,1], k = 2
Output: 2
```

Example 2:

```
Input: nums = [1,2,3], k = 3
Output: 2
```

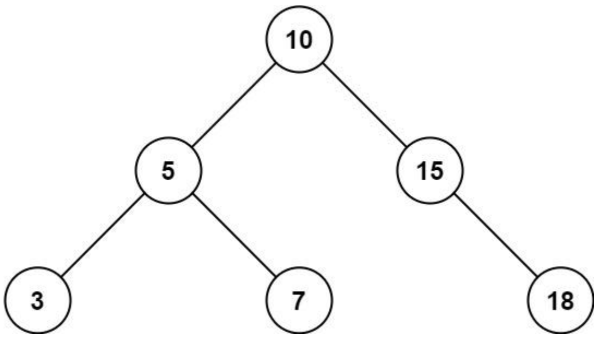
```
public int subarraySum(int[] nums, int k) {
        int answer = 0;

        Map<Integer, Integer> count = new HashMap<Integer, Integer>();

        count.put(0,1);

        int prefixSum = 0;
```

```
            for(int num : nums)
            {
                prefixSum += num;
                answer += count.getOrDefault(prefixSum-k, 0);
                count.put(prefixSum, count.getOrDefault(prefixSum, 0) + 1);
            }
            return answer;
        }

    TC: O(n)
    SC: O(n)
```

43. Range Sum of BST

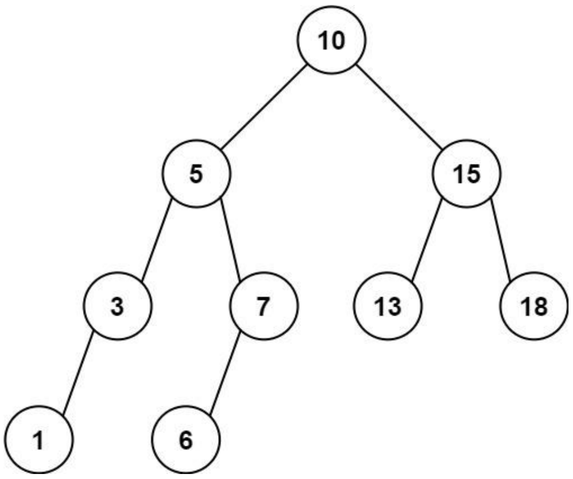
Given the root node of a binary search tree and two integers low and high, return *the sum of values of all nodes with a value in the inclusive range* [low, high].

Example 1:



```
Input: root = [10,5,15,3,7,null,18], low = 7, high = 15
Output: 32
Explanation: Nodes 7, 10, and 15 are in the range [7, 15]. 7 + 10 + 15 = 32.
```

Example 2:



```
Input: root = [10,5,15,3,7,13,18,1,null,6], low = 6, high = 10
Output: 23
Explanation: Nodes 6, 7, and 10 are in the range [6, 10]. 6 + 7 + 10 = 23.
```

```
    class Solution {
        private int sum = 0;
        public int rangeSumBST(TreeNode root, int low, int high) {
            dfs(root, low, high);
            return this.sum;
        }

        private void dfs(TreeNode root, int low, int high)
        {
            if(root != null)
            {
                if(root.val>= low && root.val<=high)
                    this.sum+=root.val;
                if(root.val>low)
                    dfs(root.left, low, high);
                if(root.val<high)
                    dfs(root.right, low, high);
            }
        }

    TC: O(n)
    SC: O(n)
```

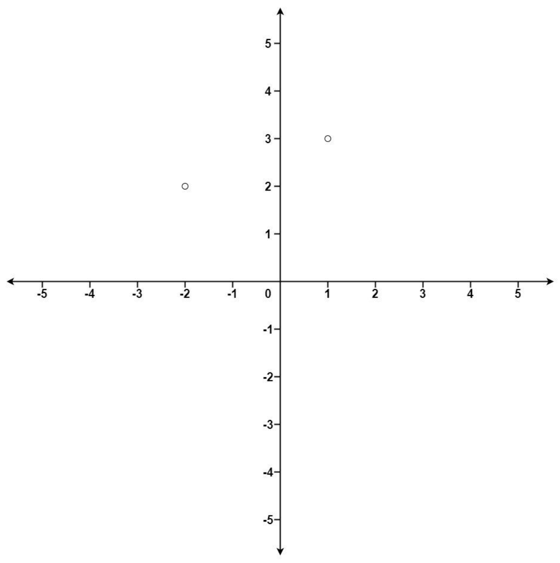
44. K Closest Points to Origin

Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0).

The distance between two points on the X-Y plane is the Euclidean distance (i.e., √(x1 - x2)2 + (y1 - y2)2).

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in).

Example 1:



```
Input: points = [[1,3],[-2,2]], k = 1
Output: [[-2,2]]
Explanation:
The distance between (1, 3) and the origin is sqrt(10).
The distance between (-2, 2) and the origin is sqrt(8).
Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.
We only want the closest k = 1 points from the origin, so the answer is just [[-2,2]].
```

Example 2:

```
Input: points = [[3,3],[5,-1],[-2,4]], k = 2
Output: [[3,3],[-2,4]]
Explanation: The answer [[-2,4],[3,3]] would also be accepted.
```

```
public int[][] KClosest(int[][] points, int k)
    {
        int n = points.GetLength(0);

        int[] distance = new int[n];

        for (int i = 0; i < n; i++)
        {
            int x = points[i][0],
                y = points[i][1];
            distance[i] = (x * x) +
                          (y * y);
        }

        Array.Sort(distance);

        // Find the k-th distance
        int distk = distance[k - 1];

        // Print all distances which are
        // smaller than k-th distance
        int [][]result = new int[n][];
        int index = 0;
        for (int i = 0; i < n; i++)
        {
            int x = points[i][0],
                y = points[i][1];
            int dist = (x * x) +
                       (y * y);

            if (dist <= distk)
            {
                result[index] = new int[2];
                result[index][0] = x;
                result[index][1] = y;
                index++;
            }
        }
        return result.Where(x => x!=null).ToArray();
    }

TC: O(nlogn)
SC: O(n)
```

45. Robot Bounded In Circle

On an infinite plane, a robot initially stands at `(0, 0)` and faces north. Note that:

- The north direction is the positive direction of the y-axis.

- The south direction is the negative direction of the y-axis.
- The east direction is the positive direction of the x-axis.
- The west direction is the negative direction of the x-axis.

The robot can receive one of three instructions:

- "G": go straight 1 unit.
- "L": turn 90 degrees to the left (i.e., anti-clockwise direction).
- "R": turn 90 degrees to the right (i.e., clockwise direction).

The robot performs the instructions given in order, and repeats them forever.

Return true if and only if there exists a circle in the plane such that the robot never leaves the circle.

Example 1:

```
Input: instructions = "GGLLGG"
Output: true
Explanation: The robot is initially at (0, 0) facing the north direction.
"G": move one step. Position: (0, 1). Direction: North.
"G": move one step. Position: (0, 2). Direction: North.
"L": turn 90 degrees anti-clockwise. Position: (0, 2). Direction: West.
"L": turn 90 degrees anti-clockwise. Position: (0, 2). Direction: South.
"G": move one step. Position: (0, 1). Direction: South.
"G": move one step. Position: (0, 0). Direction: South.
Repeating the instructions, the robot goes into the cycle: (0, 0) --> (0, 1) --> (0, 2) --> (0, 1) --> (0,
0).
Based on that, we return true.
```

Example 2:

```
Input: instructions = "GG"
Output: false
Explanation: The robot is initially at (0, 0) facing the north direction.
"G": move one step. Position: (0, 1). Direction: North.
"G": move one step. Position: (0, 2). Direction: North.
Repeating the instructions, keeps advancing in the north direction and does not go into cycles.
Based on that, we return false.
```

Example 3:

```
Input: instructions = "GL"
Output: true
Explanation: The robot is initially at (0, 0) facing the north direction.
"G": move one step. Position: (0, 1). Direction: North.
"L": turn 90 degrees anti-clockwise. Position: (0, 1). Direction: West.
"G": move one step. Position: (-1, 1). Direction: West.
"L": turn 90 degrees anti-clockwise. Position: (-1, 1). Direction: South.
"G": move one step. Position: (-1, 0). Direction: South.
"L": turn 90 degrees anti-clockwise. Position: (-1, 0). Direction: East.
"G": move one step. Position: (0, 0). Direction: East.
"L": turn 90 degrees anti-clockwise. Position: (0, 0). Direction: North.
Repeating the instructions, the robot goes into the cycle: (0, 0) --> (0, 1) --> (-1, 1) --> (-1, 0) --> (0,
0).
Based on that, we return true.
```

```
public bool IsRobotBounded(string instructions)
        {
            char direction = 'N';
    int x = 0;
    int y = 0;
    foreach(char c in instructions) {
        if (c == 'G') {
            if (direction == 'N') {
                y++;
            } else if (direction == 'S') {
                y--;
            } else if (direction == 'W') {
                x--;
            } else {
                x++;
            }
        } else if (c == 'L') {
            if (direction == 'N') {
                direction = 'W';
            } else if (direction == 'S') {
                direction = 'E';
            } else if (direction == 'W') {
                direction = 'S';
            } else {
                direction = 'N';
            }
        } else {
            if (direction == 'N') {
                direction = 'E';
            } else if (direction == 'S') {
                direction = 'W';
            } else if (direction == 'W') {
                direction = 'N';
            } else {
                direction = 'S';
            }
        }
    }
    return (x == 0 && y == 0) || (direction != 'N'); // direction would never be N if coordinates
are not (0, 0)
        }

TC: O(n)
SC: O(1)
```

46. Longest Common Subsequence

Given two strings text1 and text2, return *the length of their longest common subsequence*. If there is no common subsequence, return 0.

A subsequence of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, "ace" is a subsequence of "abcde".

A common subsequence of two strings is a subsequence that is common to both strings.

Example 1:

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.
```

Example 2:

```
Input: text1 = "abc", text2 = "abc"
Output: 3
Explanation: The longest common subsequence is "abc" and its length is 3.
```

Example 3:

```
Input: text1 = "abc", text2 = "def"
Output: 0
Explanation: There is no such common subsequence, so the result is 0.
```

```
public int longestCommonSubsequence(String text1, String text2) {

    // Make a grid of 0's with text2.length() + 1 columns
    // and text1.length() + 1 rows.
    int[][] dpGrid = new int[text1.length() + 1][text2.length() + 1];

    // Iterate up each column, starting from the last one.
    for (int col = text2.length() - 1; col >= 0; col--) {
      for (int row = text1.length() - 1; row >= 0; row--) {
        // If the corresponding characters for this cell are the same...
        if (text1.charAt(row) == text2.charAt(col)) {
          dpGrid[row][col] = 1 + dpGrid[row + 1][col + 1];
        // Otherwise they must be different...
        } else {
          dpGrid[row][col] = Math.max(dpGrid[row + 1][col], dpGrid[row][col + 1]);
        }
      }
    }

    // The original problem's answer is in dp_grid[0][0]. Return it.
    return dpGrid[0][0];
  }

TC: O(n*m)
SC: O(n*m)
```

47. Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts

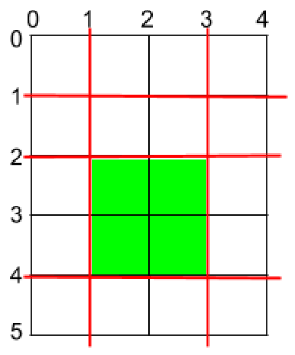
You are given a rectangular cake of size h x w and two arrays of integers horizontalCuts and verticalCuts where:

- horizontalCuts[i] is the distance from the top of the rectangular cake to the ith horizontal cut and similarly, and
- verticalCuts[j] is the distance from the left of the rectangular cake to the jth vertical cut.

Return *the maximum area of a piece of cake after you cut at each horizontal and vertical position provided in the arrays* horizontalCuts *and* verticalCuts.

Since the answer can be a large number, return this modulo 109 + 7.

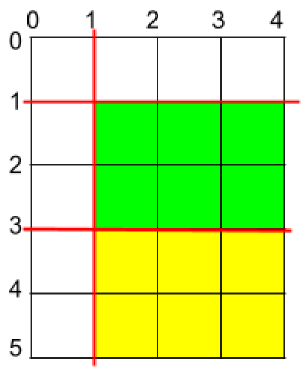
Example 1:



```
Input: h = 5, w = 4, horizontalCuts = [1,2,4], verticalCuts = [1,3]
```

```
Output: 4
Explanation: The figure above represents the given rectangular cake. Red lines are the horizontal and
vertical cuts. After you cut the cake, the green piece of cake has the maximum area.
```

Example 2:



```
Input: h = 5, w = 4, horizontalCuts = [3,1], verticalCuts = [1]
Output: 6
Explanation: The figure above represents the given rectangular cake. Red lines are the horizontal and
vertical cuts. After you cut the cake, the green and yellow pieces of cake have the maximum area.
```

Example 3:

```
Input: h = 5, w = 4, horizontalCuts = [3], verticalCuts = [3]
Output: 9
```

```
public int MaxArea(int h, int w, int[] horizontalCuts, int[] verticalCuts)
        {
            Array.Sort(horizontalCuts);
            Array.Sort(verticalCuts);

            //boundary condition
            long maxH = Math.Max(horizontalCuts[0], h - horizontalCuts[horizontalCuts.Length-1]);

            for(int i=1; i<horizontalCuts.Length; i++)
            {
                maxH = Math.Max(horizontalCuts[i] - horizontalCuts[i - 1], maxH);
            }

            //boundary condition
            long maxV = Math.Max(verticalCuts[0], w - verticalCuts[verticalCuts.Length-1]);

            for (int i = 1; i < verticalCuts.Length; i++)
            {
                maxV = Math.Max(verticalCuts[i] - verticalCuts[i - 1], maxV);
            }

            return Convert.ToInt32((maxH * maxV)%(1000000007));
        }

TC: O(nlogn)
SC: O(logn)
```

48. Dot Product of Two Sparse Vectors

Given two sparse vectors, compute their dot product.

Implement class `SparseVector`:

- `SparseVector(nums)` Initializes the object with the vector `nums`

- `dotProduct(vec)` Compute the dot product between the instance of *SparseVector* and `vec`

A sparse vector is a vector that has mostly zero values, you should store the sparse vector efficiently and compute the dot product between two *SparseVector*.

Follow up: What if only one of the vectors is sparse?

Example 1:

```
Input: nums1 = [1,0,0,2,3], nums2 = [0,3,0,4,0]
Output: 8
Explanation: v1 = SparseVector(nums1) , v2 = SparseVector(nums2)
v1.dotProduct(v2) = 1*0 + 0*3 + 0*0 + 2*4 + 3*0 = 8
```

Example 2:

```
Input: nums1 = [0,1,0,0,0], nums2 = [0,0,0,0,2]
Output: 0
Explanation: v1 = SparseVector(nums1) , v2 = SparseVector(nums2)
v1.dotProduct(v2) = 0*0 + 1*0 + 0*0 + 0*0 + 0*2 = 0
```

Example 3:

```
Input: nums1 = [0,1,0,0,2,0,0], nums2 = [1,0,0,0,3,0,4]
Output: 6
```

```
public class SparseVector {

    public Dictionary<int, int> keyValuePairs = new Dictionary<int, int>();
        public SparseVector(int[] nums)
        {
            for(int i=0;i<nums.Length;i++)
            {
                if (nums[i] != 0)
                    keyValuePairs.Add(i, nums[i]);
            }
        }

        // Return the dotProduct of two sparse vectors
        public int DotProduct(SparseVector vec)
        {
            int result = 0;
            foreach(int key in this.keyValuePairs.Keys)
            {
                if(vec.keyValuePairs.ContainsKey(key))
                    result += this.keyValuePairs[key] * vec.keyValuePairs[key];
            }
            return result;
        }
}

TC: O(n)
SC: O(n)
```

49. Buildings With an Ocean View

There are n buildings in a line. You are given an integer array heights of size n that represents the heights of the buildings in the line.

The ocean is to the right of the buildings. A building has an ocean view if the building can see the ocean without obstructions. Formally, a building has an ocean view if all the buildings to its right have a smaller height.

Return a list of indices (0-indexed) of buildings that have an ocean view, sorted in increasing order.

Example 1:

```
Input: heights = [4,2,3,1]
Output: [0,2,3]
Explanation: Building 1 (0-indexed) does not have an ocean view because building 2 is taller.
```

Example 2:

```
Input: heights = [4,3,2,1]
Output: [0,1,2,3]
Explanation: All the buildings have an ocean view.
```

Example 3:

```
Input: heights = [1,3,2,4]
Output: [3]
Explanation: Only building 3 has an ocean view.
```

```
Solution 1: Using Set
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        Set<ListNode> visited = new HashSet();

        while(headB != null)
        {
            visited.add(headB);
            headB = headB.next;
        }

        while(headA != null)
        {
            if(visited.contains(headA))
                return headA;
            headA = headA.next;
        }
        return null;
    }
TC: O(|V|)
SC: O(N)
V -> number of nodes
N -> length of B Node
```

```
Solution 2: Interchange pointers
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        ListNode p1 = headA;
        ListNode p2 = headB;

        while(p1 != p2)
        {
            p1 = p1 == null ? headB : p1.next;
            p2 = p2 == null ? headA : p2.next;
        }
        return p1;
    }
TC: O(N)
SC: O(1)
```

50. Longest Increasing Subsequence (Medium)

Given an integer array nums, return the length of the longest strictly increasing subsequence.

A subsequence is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements. For example, [3,6,2,7] is a subsequence of the array [0,3,1,6,2,2,7].

Example 1:

```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
```

```
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
```

Example 2:

```
Input: nums = [0,1,0,3,2,3]
Output: 4
```

Example 3:

```
Input: nums = [7,7,7,7,7,7,7]
Output: 1
```

```
public int LengthOfLIS(int[] nums) {
        int[] dp = new int[nums.Length];

        Array.Fill(dp,1);

        for(int i=1; i<nums.Length; i++)
        {
            for(int j=0; j<i; j++)
            {
                if(nums[i]>nums[j])
                    dp[i] = Math.Max(dp[i], dp[j]+1);
            }
        }

        return dp.Max();
    }
```

TC: O(n^2)
SC: O(n)

51. Longest Consecutive Sequence (Medium)

Given an unsorted array of integers nums, return *the length of the longest consecutive elements sequence.*

You must write an algorithm that runs in O(n) time.

Example 1:

```
Input: nums = [100,4,200,1,3,2]
Output: 4
```

```
Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.
```

Example 2:

```
Input: nums = [0,3,7,2,5,8,4,6,0,1]
Output: 9
```

```
 public int LongestConsecutive(int[] nums) {

         HashSet<int> set = new HashSet<int>();
         foreach(int n in nums)
         {
             set.Add(n);
         }

         int maxStreak = 0;

         foreach(int n in set)
         {
             if(!set.Contains(n-1))
             {
                 int current = n;
                 int currentStreak = 1;

                 while(set.Contains(current+1))
                 {
                     current++;
                     currentStreak++;
                 }
                 maxStreak = Math.Max(maxStreak, currentStreak);
             }
         }
     }
```

TC: O(N)

SC: O(N)

52. House Robber

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight without alerting the police*.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 4

Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = 1 + 3 = 4.
```

Example 2:

```
Input: nums = [2,7,9,3,1]

Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).

Total amount you can rob = 2 + 9 + 1 = 12.
```

```
public int Rob(int[] nums) {

        int n = nums.Length;

        if(n==0)
            return 0;

        int []dp = new int[n+1];

        dp[n] = 0;
        dp[n-1] = nums[n-1];

        for(int i = n-2; i>=0; i--)
        {
            dp[i] = Math.Max(dp[i+1], dp[i+2] + nums[i]);
        }

        return dp[0];
    }
TC: O(n)
SC: O(n)
```

53. House Robber II

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have a security system connected, and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight without alerting the police*.

Example 1:

```
Input: nums = [2,3,2]
Output: 3
```

```
Explanation: You cannot rob house 1 (money = 2) and then rob house 3 (money = 2), because they are adjacent houses.
```

Example 2:

```
Input: nums = [1,2,3,1]
Output: 4
```

```
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
```

```
Total amount you can rob = 1 + 3 = 4.
```

Example 3:

```
Input: nums = [1,2,3]
Output: 3
```

```
public int Rob(int[] nums) {
        int n = nums.Length;

        if(n == 0)
            return 0;

        if(n == 1)
            return nums[0];


        return Math.Max(RobDp(nums, 0, n-2), RobDp(nums, 1, n-1));
    }

    private int RobDp(int[] nums, int start, int end)
    {
        int t1 = 0;
        int t2 = 0;

        for(int i=start; i<=end; i++)
        {
            int temp = t1;
            t1 = Math.Max(t1, t2+nums[i]);
            t2 = temp;
        }

        return t1;
    }
```

TC: O(N)

SC: O(1)

54. Top K Frequent Elements (Medium)

Given an integer array nums and an integer k, return *the* k *most frequent elements*. You may return the answer in any order.

Example 1:

```
Input: nums = [1,1,1,2,2,3], k = 2
Output: [1,2]
```

Example 2:

```
Input: nums = [1], k = 1
Output: [1]
```

```
public int[] TopKFrequent(int[] nums, int k) {
        if (nums.Length == k)
            return nums;

            Dictionary<int, int> freqMap = new Dictionary<int, int> ();

            foreach (int n in nums)
            {
                if (freqMap.ContainsKey(n))
                    freqMap[n]++;
                else
                    freqMap.Add(n, 1);
            }

            List<IList<int>> bucket = new List<IList<int>>();

            for (int i = 0; i < nums.Length+1; i++)
            {
                bucket.Add(new List<int>());
            }

            foreach (int key in freqMap.Keys)
            {
                bucket[freqMap[key]].Add(key);
            }

            int[] ans = new int[k];

            for(int z=nums.Length; z>=0; z--)
            {
                foreach(int e in bucket[z])
                {
                    ans[k-1] = e;
                    k--;
                    if (k == 0)
                        return ans;
                }
            }

            return nums;
    }

TC: O(n)
SC: O(n)
```

55. Search in Rotated Sorted Array

There is an integer array nums sorted in ascending order (with distinct values).

Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums after the possible rotation and an integer target, return *the index of* target *if it is in* nums*, or* -1 *if it is not in* nums.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

```
Input: nums = [4,5,6,7,0,1,2], target = 0
```

```
Output: 4
```

Example 2:

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

Example 3:

```
Input: nums = [1], target = 0
Output: -1
```

```
public int Search(int[] nums, int target) {
        int start = 0;
        int end = nums.Length-1;

        while(start<=end)
        {
            int mid = (start+end)/2;

            if(nums[mid] == target)
                return mid;
            else if(nums[mid]>=nums[start])
            {
                if(target < nums[mid] && target >= nums[start])
                    end = mid - 1;
                else
                    start = mid+1;
            }
            else
            {
                if(target <= nums[end] && target > nums[mid])
                    start = mid +1;
                else
                    end = mid - 1;
            }
        }
        return -1;

    }

TC: O(logn)
SC: O(1)
```

56. Plus One

You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return *the resulting array of digits*.

Example 1:

```
Input: digits = [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
Incrementing by one gives 123 + 1 = 124.
Thus, the result should be [1,2,4].
```

Example 2:

```
Input: digits = [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
Incrementing by one gives 4321 + 1 = 4322.
Thus, the result should be [4,3,2,2].
```

Example 3:

```
Input: digits = [9]
Output: [1,0]
Explanation: The array represents the integer 9.
Incrementing by one gives 9 + 1 = 10.
Thus, the result should be [1,0].
```

```
public int[] PlusOne(int[] digits) {
    int n = digits.Length;

    // move along the input array starting from the end
    for (int idx = n - 1; idx >= 0; --idx) {
      // set all the nines at the end of array to zeros
      if (digits[idx] == 9) {
        digits[idx] = 0;
      }
      // here we have the rightmost not-nine
      else {
        // increase this rightmost not-nine by 1
        digits[idx]++;
        // and the job is done
        return digits;
      }
    }
    // we're here because all the digits are nines
    digits = new int[n + 1];
    digits[0] = 1; //By default all other elements would be 0
    return digits;
  }

TC: O(n)
SC: O(1)
```

57. Decode Ways

A message containing letters from A-Z can be encoded into numbers using the following mapping:

```
'A' -> "1"
'B' -> "2"
...
'Z' -> "26"
```

To decode an encoded message, all the digits must be grouped then mapped back into letters using the reverse of the mapping above (there may be multiple ways). For example, "11106" can be mapped into:

- "AAJF" with the grouping (1 1 10 6)
- "KJF" with the grouping (11 10 6)

Note that the grouping (1 11 06) is invalid because "06" cannot be mapped into 'F' since "6" is different from "06".

Given a string s containing only digits, return *the number of ways to decode it*.

The test cases are generated so that the answer fits in a 32-bit integer.

Example 1:

```
Input: s = "12"
Output: 2
Explanation: "12" could be decoded as "AB" (1 2) or "L" (12).
```

Example 2:

```
Input: s = "226"
Output: 3
Explanation: "226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).
```

Example 3:

```
Input: s = "06"
Output: 0
Explanation: "06" cannot be mapped to "F" because of the leading zero ("6" is different from "06").
```

```
public int NumDecodings(string s) {
        int n = s.Length;

        if(s[0] == '0')
            return 0;

        int twoBack = 1;
        int oneBack = 1;

        for(int i=1; i<n;  i++)
        {
            int current = 0;
            if(s[i] != '0')
                current += oneBack;

            int twoDigit = int.Parse(s.Substring(i-1, 2));

            if(twoDigit>=10 && twoDigit<=26)
                current += twoBack;

            twoBack = oneBack;
            oneBack = current;
        }

        return oneBack;
    }

TC: O(n)
SC: O(1)
```

58. Clone Graph

Given a reference of a node in a connected undirected graph.

Return a deep copy (clone) of the graph.

Each node in the graph contains a value (`int`) and a list (`List[Node]`) of its neighbors.

```
class Node {
    public int val;
    public List<Node> neighbors;
}
```

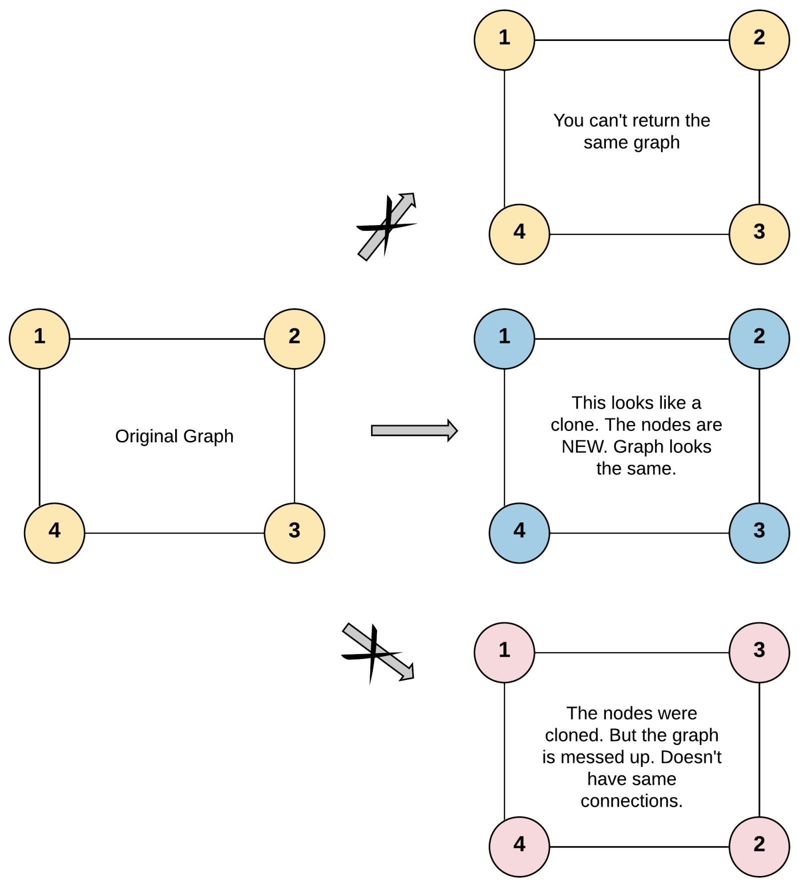
Test case format:

For simplicity, each node's value is the same as the node's index (1-indexed). For example, the first node with `val == 1`, the second node with `val == 2`, and so on. The graph is represented in the test case using an adjacency list.

An adjacency list is a collection of unordered lists used to represent a finite graph. Each list describes the set of neighbors of a node in the graph.

The given node will always be the first node with `val = 1`. You must return the copy of the given node as a reference to the cloned graph.

Example 1:


You can't return the same graph
Original Graph
This looks like a clone. The nodes are NEW. Graph looks the same.
The nodes were cloned. But the graph is messed up. Doesn't have same connections.

```
Input: adjList = [[2,4],[1,3],[2,4],[1,3]]
Output: [[2,4],[1,3],[2,4],[1,3]]
Explanation: There are 4 nodes in the graph.
1st node (val = 1)'s neighbors are 2nd node (val = 2) and 4th node (val = 4).
2nd node (val = 2)'s neighbors are 1st node (val = 1) and 3rd node (val = 3).
3rd node (val = 3)'s neighbors are 2nd node (val = 2) and 4th node (val = 4).
4th node (val = 4)'s neighbors are 1st node (val = 1) and 3rd node (val = 3).
```

Example 2:

```
Input: adjList = [[]]
Output: [[]]
Explanation: Note that the input contains one empty list. The graph consists of only one node
with val = 1 and it does not have any neighbors.
```

Example 3:

```
Input: adjList = []
```

```
Output: []
Explanation: This an empty graph, it does not have any nodes.
```

```
/*
// Definition for a Node.
public class Node {
    public int val;
    public IList<Node> neighbors;

    public Node() {
        val = 0;
        neighbors = new List<Node>();
    }

    public Node(int _val) {
        val = _val;
        neighbors = new List<Node>();
    }

    public Node(int _val, List<Node> _neighbors) {
        val = _val;
        neighbors = _neighbors;
    }
}
*/

public class Solution {
    Dictionary<Node, Node> visited = new Dictionary<Node, Node>();
    public Node CloneGraph(Node node) {
        if(node == null)
            return node;

        if(visited.ContainsKey(node))
            return visited[node];

        Node cloneNode = new Node(node.val, new List<Node>());
        visited.Add(node, cloneNode);

        foreach(Node n in node.neighbors)
        {
            cloneNode.neighbors.Add(CloneGraph(n));
        }

        return cloneNode;
    }
}

TC: O(|E|+|V|)
SC: O(|V|)
```

59. Word Break

Given a string s and a dictionary of strings wordDict, return true if s can be segmented into a space-separated sequence of one or more dictionary words.

Note that the same word in the dictionary may be reused multiple times in the segmentation.

Example 1:

```
Input: s = "leetcode", wordDict = ["leet","code"]
Output: true
Explanation: Return true because "leetcode" can be segmented as "leet code".
```

Example 2:

```
Input: s = "applepenapple", wordDict = ["apple","pen"]
Output: true
Explanation: Return true because "applepenapple" can be segmented as "apple pen apple".
Note that you are allowed to reuse a dictionary word.
```

Example 3:

```
Input: s = "catsandog", wordDict = ["cats","dog","sand","and","cat"]
Output: false
```

```
public bool WordBreak(string s, IList<string> wordDict) {
        HashSet<string> set = new HashSet<string>(wordDict);

        int n = s.Length;
        bool []dp = new bool[n+1];

        dp[0] = true;

        for(int i=1; i<=n; i++)
        {
            for(int j=0; j<i; j++)
            {
                if(dp[j] && set.Contains(s.Substring(j, i-j)))
                {
                    dp[i] = true;
                    break;
                }
            }
        }
        return dp[n];
    }
```

TC: O(n^2)
SC: O(n)

60. Find Minimum in Rotated Sorted Array

Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

- [4,5,6,7,0,1,2] if it was rotated 4 times.
- [0,1,2,4,5,6,7] if it was rotated 7 times.

Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums of unique elements, return *the minimum element of this array*.

You must write an algorithm that runs in O(log n) time.

Example 1:

```
Input: nums = [3,4,5,1,2]
Output: 1
Explanation: The original array was [1,2,3,4,5] rotated 3 times.
```

Example 2:

```
Input: nums = [4,5,6,7,0,1,2]
Output: 0
Explanation: The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.
```

Example 3:

```
Input: nums = [11,13,15,17]
Output: 11
Explanation: The original array was [11,13,15,17] and it was rotated 4 times.
```

```
public int FindMin(int[] nums) {

        int length = nums.Length;
        if(length == 1)
            return nums[0];

        if(nums[0]<nums[length-1])
            return nums[0];

        int left = 0;
        int right = length-1;

        while(left<=right)
        {
            int mid = left + (right-left)/2;

            if(nums[mid]>nums[mid+1])
                return nums[mid+1];

            if(nums[mid-1]>nums[mid])
                return nums[mid];

            if(nums[mid]>nums[0])
                left = mid+1;
            else
                right = mid-1;
        }
        return -1;
    }

TC: O(logn)
SC: O(1)
```

61. Missing Number

Given an array nums containing n distinct numbers in the range [0, n], return *the only number in the range that is missing from the array.*

Example 1:

```
Input: nums = [3,0,1]
Output: 2
Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the
range since it does not appear in nums.
```

Example 2:

```
Input: nums = [0,1]
Output: 2
Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the
range since it does not appear in nums.
```

Example 3:

```
Input: nums = [9,6,4,2,3,5,7,0,1]
Output: 8
Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the
range since it does not appear in nums.
```

```
public int missingNumber(int[] nums) {
        int expectedSum=nums.length*(nums.length+1)/2;
        int actualSum=0;
        for(int num: nums){
            actualSum+=num;
        }
        return expectedSum-actualSum;
    }
TC: O(n)
SC: O(1)
```

62. Sum of Two Integers

Given two integers a and b, return *the sum of the two integers without using the operators* + *and* -.

Example 1:

```
Input: a = 1, b = 2
Output: 3
```

Example 2:

```
Input: a = 2, b = 3
Output: 5
```

```
public int GetSum(int a, int b) {
        while (b != 0) {
            int answer = a ^ b;
```

```
                int carry = (a & b) << 1;
                a = answer;
                b = carry;
            }

            return a;
        }
```

63. Combination Sum IV

Given an array of distinct integers `nums` and a target integer `target`, return *the number of possible combinations that add up to* `target`.

The test cases are generated so that the answer can fit in a 32-bit integer.

Example 1:

```
Input: nums = [1,2,3], target = 4
Output: 7
Explanation:
The possible combination ways are:
(1, 1, 1, 1)
(1, 1, 2)
(1, 2, 1)
(1, 3)
(2, 1, 1)
(2, 2)
(3, 1)
Note that different sequences are counted as different combinations.
```

Example 2:

```
Input: nums = [9], target = 3
Output: 0
```

```
    public int CombinationSum4(int[] nums, int target) {
            int []dp = new int[target+1];

            dp[0] = 1;

            for(int i=1; i<=target; i++)
            {
                foreach(int num in nums)
                {
                    if(i-num >=0)
                        dp[i] += dp[i-num];
                }
            }
            return dp[target];
        }
    TC: O(n*t)
    SC: O(t)
    t -> target
```

64. Pacific Atlantic Water Flow

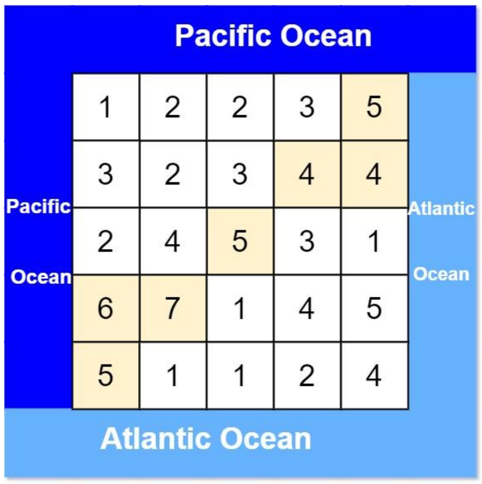
There is an m x n rectangular island that borders both the Pacific Ocean and Atlantic Ocean. The Pacific Ocean touches the island's left and top edges, and the Atlantic Ocean touches the island's right and bottom edges.

The island is partitioned into a grid of square cells. You are given an m x n integer matrix heights where heights[r][c] represents the height above sea level of the cell at coordinate (r, c).

The island receives a lot of rain, and the rain water can flow to neighboring cells directly north, south, east, and west if the neighboring cell's height is less than or equal to the current cell's height. Water can flow from any cell adjacent to an ocean into the ocean.

Return *a 2D list of grid coordinates* result *where* result[i] = [ri, ci] *denotes that rain water can flow from cell* (ri, ci) *to both the Pacific and Atlantic oceans*.

Example 1:



```
Input: heights = [[1,2,2,3,5],[3,2,3,4,4],[2,4,5,3,1],[6,7,1,4,5],[5,1,1,2,4]]
Output: [[0,4],[1,3],[1,4],[2,2],[3,0],[3,1],[4,0]]
Explanation: The following cells can flow to the Pacific and Atlantic oceans, as shown below:
[0,4]: [0,4] -> Pacific Ocean
       [0,4] -> Atlantic Ocean
[1,3]: [1,3] -> [0,3] -> Pacific Ocean
       [1,3] -> [1,4] -> Atlantic Ocean
[1,4]: [1,4] -> [1,3] -> [0,3] -> Pacific Ocean
       [1,4] -> Atlantic Ocean
[2,2]: [2,2] -> [1,2] -> [0,2] -> Pacific Ocean
       [2,2] -> [2,3] -> [2,4] -> Atlantic Ocean
[3,0]: [3,0] -> Pacific Ocean
       [3,0] -> [4,0] -> Atlantic Ocean
[3,1]: [3,1] -> [3,0] -> Pacific Ocean
       [3,1] -> [4,1] -> Atlantic Ocean
[4,0]: [4,0] -> Pacific Ocean
       [4,0] -> Atlantic Ocean
Note that there are other possible paths for these cells to flow to the Pacific and Atlantic oceans.
```

Example 2:

```
Input: heights = [[1]]
Output: [[0,0]]
Explanation: The water can flow from the only cell to the Pacific and Atlantic oceans.
```

```
Solution 1: Using DFS
public class Solution {

    private int numRows;
    private int numCols;
    private int [][]heights;
    private int[][] directions = new int[][]{new int[]{0,1}, new int[]{1,0}, new int[]{-1, 0},
new int[]{0, -1}};

    public IList<IList<int>> PacificAtlantic(int[][] heights) {
        numRows = heights.Length;
        numCols = heights[0].Length;
        this.heights = heights;

        bool [,]pacificReach = new bool[numRows, numCols];
        bool [,]atlanticReach = new bool[numRows, numCols];

        for(int i=0; i<numRows; i++)
        {
            dfs(i, 0, pacificReach);
            dfs(i, numCols-1, atlanticReach);
        }

        for(int i=0; i<numCols; i++)
        {
            dfs(0, i, pacificReach);
            dfs(numRows-1, i, atlanticReach);
        }
        IList<IList<int>> commons = new List<IList<int>>();
        for(int i=0; i<numRows; i++)
        {
            for(int j=0; j<numCols; j++)
            {
                if(pacificReach[i,j] && atlanticReach[i,j])
                    commons.Add(new List<int>(){i, j});
            }
        }
        return commons;
    }
    private void dfs(int r, int c, bool [,]reachable)
    {
        reachable[r, c] = true;
        foreach(int []dir in directions)
        {
            int newRow = r + dir[0];
            int newCol = c + dir[1];

            if(newRow < 0 || newRow >= numRows || newCol < 0 || newCol >= numCols ||
reachable[newRow, newCol])
                continue;

            if(heights[newRow][newCol] < heights[r][c] )
                continue;

            dfs(newRow, newCol, reachable);
        }
    }
}
```

```
Solution 2: Using BFS
public class Solution {

    private int numRows;
    private int numCols;
    private int [][]heights;
    private int[][] directions = new int[][]{new int[]{0,1}, new int[]{1,0}, new int[]{-1, 0},
new int[]{0, -1}};

    public IList<IList<int>> PacificAtlantic(int[][] heights) {
        numRows = heights.Length;
        numCols = heights[0].Length;
        this.heights = heights;

        Queue<int[]> pacificQueue = new Queue<int[]>();
        Queue<int[]> atlanticQueue = new Queue<int[]>();

        for(int i=0; i<numRows; i++)
        {
            pacificQueue.Enqueue(new int[]{i,0});
            atlanticQueue.Enqueue(new int[]{i, numCols-1});
        }

        for(int i=0; i<numCols; i++)
        {
            pacificQueue.Enqueue(new int[]{0,i});
```

```
                atlanticQueue.Enqueue(new int[]{numRows-1, i});
            }

            bool[,] pacificReach = bfs(pacificQueue);
            bool[,] atlanticReach = bfs(atlanticQueue);

            IList<IList<int>> commons = new List<IList<int>>();

            for(int i=0; i<numRows; i++)
            {
                for(int j=0; j<numCols; j++)
                {
                    if(pacificReach[i,j] && atlanticReach[i,j])
                        commons.Add(new List<int>(){i, j});
                }
            }

            return commons;
        }
```

```
    private bool[,] bfs(Queue<int[]> q)
        {
            bool [,]reachable = new bool[numRows, numCols];
            while(q.Count != 0)
            {
                int[] val = q.Dequeue();
                reachable[val[0], val[1]] = true;

                foreach(int[] dir in directions)
                {
                    int newRow = val[0] + dir[0];
                    int newCol = val[1] + dir[1];

                    if(newRow < 0 || newRow >= numRows || newCol < 0 || newCol >= numCols ||
    reachable[newRow, newCol])
                        continue;

                    if(heights[newRow][newCol] < heights[val[0]][val[1]])
                        continue;

                    q.Enqueue(new int[]{newRow, newCol});
                }
            }
            return reachable;
        }
    }
```

65. Maximum Points You Can Obtain from Cards

There are several cards arranged in a row, and each card has an associated number of points. The points are given in the integer array cardPoints.

In one step, you can take one card from the beginning or from the end of the row. You have to take exactly k cards.

Your score is the sum of the points of the cards you have taken.

Given the integer array cardPoints and the integer k, return the *maximum score* you can obtain.

Example 1:

```
Input: cardPoints = [1,2,3,4,5,6,1], k = 3
Output: 12
Explanation: After the first step, your score will always be 1. However, choosing the rightmost card first will maximize
your total score. The optimal strategy is to take the three cards on the right, giving a final score of 1 + 6 + 5 = 12.
```

Example 2:

```
Input: cardPoints = [2,2,2], k = 2
Output: 4
Explanation: Regardless of which two cards you take, your score will always be 4.
```

Example 3:

```
Input: cardPoints = [9,7,7,9,7,7,9], k = 7
Output: 55
Explanation: You have to take all the cards. Your score is the sum of points of all cards.
```

```
public int MaxScore(int[] cardPoints, int k) {
        int[] front = new int[k+1];
        int[] rear = new int[k+1];
        int n = cardPoints.Length;

        for(int i=0; i<k; i++){
            front[i+1] = front[i] + cardPoints[i];
            rear[i+1] = rear[i] + cardPoints[n-1-i];
        }

        int maxScore = 0;

        for(int i=0; i<=k; i++){
            int currScore = front[i] + rear[k-i];
            maxScore = Math.Max(maxScore, currScore);
        }
        return maxScore;
    }
TC: O(k)
SC: O(k)
```