**37. Product of Array Except Self ( Input:** nums = [1,2,3,4]

**Output:** [24,12,8,6]**)**

***Arrays***

**(The idea is to: first round to calculate the product of all left digits, second round to calculate the product of all right digits.)**

**LOGIC C#**

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

if(nums.Length == 0 || nums == null)

return null;

int temp = 1;

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

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

result[i] = temp;

temp \*= nums[i];

}

temp = 1;

for(int i = nums.Length-1; i>= 0; i--){

result[i] \*= temp;

temp \*= nums[i];

}

return result;

}

**LOGIC JS**

var productExceptSelf = function(nums) {

if(nums.length == 0 || nums == null)

return null;

let temp = 1;

let result = [];

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

{

result[i] = temp;

temp \*=nums[i];

}

temp =1;

for(let i = nums.length-1; i>= 0 ; i--){

result[i] \*= temp;

temp \*= nums[i];

}

return result;

};

**38. Find the Duplicate Number(Input:** nums = [1,3,4,2,2], **Output:** 2**)**

***HashSet***

**LOGIC C#**

public int FindDuplicate(int[] nums) {

var data = new HashSet<int>();

foreach (var n in nums){

if(!set.Contains(n){

set.Add(n);

}

Else{

Return n;

}

}

Return -1;

}

**LOGIC JS**

var findDuplicate = function(nums) {

let data = new Set();

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

if(!data.has(nums[i])){

data.add(nums[i]);

}

else

return nums[i];

}

return -1;

};

**39. Find All Duplicates in an Array( Input:** nums = [4,3,2,7,8,2,3,1]

**Output:** [2,3]**)**

***Hash Set***

**LOGIC C#**

public IList<int> FindDuplicates(int[] nums) {

var result = new List<int>();

var data = new HashSet<int>();

foreach(var n in nums){

if(!data.Contains(n)){

data.Add(n);

}

Else{

Result.Add(n);

}

}

Return result;

}

**LOGIC JS**

var findDuplicates = function(nums) {

let data = new Set();

let result = [];

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

if(!data.has(nums[i])){

data.add(nums[i]);

}

else{

result.push(nums[i]);

}

}

return result;

};

**40. Set Matrix Zero (Input:** matrix = [[1,1,1],[1,0,1],[1,1,1]]

**Output:** [[1,0,1],[0,0,0],[1,0,1]]

**)**

***Arrays***

**LOGIC C#**

public void SetZeroes(int[][] matrix) {

int rows = matrix.GetLength(0);

int colums = matrix[0].GetLength(0);

bool[] markedRows = new bool[rows];

bool[] markedColums = new bool[colums];

for(int I = 0; i<rows.Length; i++){

for(int j = 0; j<colums.Length; j++){

if(matrix[i][j] == 0){

markedRows [i] = true;

markedColums[i] = true;

}

}

}

for(int i = 0; i < markedRows.Length; i++)

{

if (markedRows[i])

{

for(int j = 0; j < cols; j++)

{

matrix[i][j] = 0;

}

}

}

for(int i = 0; i < markedCols.Length; i++)

{

if (markedCols[i])

{

for(int j = 0; j < rows; j++)

{

matrix[j][i] = 0;

}

}

} }

**LOGIC JS**

var setZeroes = function(matrix) {

let locations = [];

for(let i = 0; i < matrix.length; i++) {

for(let j = 0; j < matrix[0].length; j++) {

if(matrix[i][j] == 0) {

locations.push([i, j]);

}

}

}

const setZeroWithCoordinates = (row, col) => {

for(let col1 = 0; col1 < matrix[0].length; col1++) {

matrix[row][col1] = 0;

}

for(let row1 = 0; row1 < matrix.length; row1++) {

matrix[row1][col] = 0;

}

};

for(let loc of locations) {

setZeroWithCoordinates(loc[0], loc[1])

}

};

**41.Spiral Matrix(Input:** matrix = [[1,2,3],[4,5,6],[7,8,9]]

**Output:** [1,2,3,6,9,8,7,4,5]**)**

***Arrays***

**LOGIC C#**

public IList<int> SpiralOrder(int[][] matrix) {

List<int> result = new List<int>();

if (matrix == null || matrix.Length == 0)

{

return result;

}

int direction = 0;

int top = 0, down = matrix.Length - 1, left = 0, right = matrix[0].Length - 1;

while(top<=down && left<= right)

{

if(direction == 0)

{

for(int i = left; i<= right; i++)

{

result.Add(matrix[top][i]);

}

top += 1;

}

else if(direction == 1)

{

for (int i = top; i <= down; i++)

{

result.Add(matrix[i][right]);

}

right -= 1;

}

else if (direction == 2)

{

for (int i = right; i >= left; i--)

{

result.Add(matrix[down][i]);

}

down -= 1;

}

else if (direction == 3)

{

for (int i = down; i >= top; i--)

{

result.Add(matrix[i][left]);

}

left += 1;

}

direction = (direction + 1) % 4;

}

return result;

}

**LOGIC JS**

var spiralOrder = function(matrix) {

let result = [];

let direction = 0;

let top = 0, down = matrix.length-1,left = 0; right = matrix[0].length-1;

while(top<= down && left<= right){

if(direction == 0){

for(let i =left; i<= right; i++ ){

result.push(matrix[top][i]);

}

top+=1;

}

if(direction == 1){

for(let i = top; i<= down; i++){

result.push(matrix[i][right]);

}

right-=1;

}

if(direction == 2){

for(let i =right; i>= left; i-- ){

result.push(matrix[down][i])

}

down -=1;

}

if(direction == 3){

for(let i = down; i>= top; i--){

result.push(matrix[i][left])

}

left+=1;

}

direction = (direction +1) %4;

}

return result;

};

**42. Rotate Matrix (Input:** matrix = [[1,2,3],[4,5,6],[7,8,9]]

**Output:** [[7,4,1],[8,5,2],[9,6,3]]**)**

***ARRAYS***

**LOGIC C#**

public void Rotate(int[][] matrix) {

for(int i = 0; i<matrix.Length; i++)

{

for (int j = i; j < matrix[i].Length; j++)

{

int temp = matrix[i][j];

matrix[i][j] = matrix[j][i];

matrix[j][i] = temp;

}

}

for(int k = 0; k<matrix.Length; k++)

{

int start = 0;

int end = matrix[k].Length - 1;

while (start < end)

{

int temp = matrix[k][start];

matrix[k][start] = matrix[k][end];

matrix[k][end] = temp;

start++;

end--;

}

}

}

**LOGIC JS**

var rotate = function(matrix) {

for(let i = 0; i<matrix.length; i++)

{

for (let j = i; j < matrix[i].length; j++)

{

let temp = matrix[i][j];

matrix[i][j] = matrix[j][i];

matrix[j][i] = temp;

}

}

for(let k = 0; k<matrix.length; k++)

{

let start = 0;

let end = matrix[k].length - 1;

while (start < end)

{

let temp = matrix[k][start];

matrix[k][start] = matrix[k][end];

matrix[k][end] = temp;

start++;

end--;

}

}

};

**43.Word Search(Input:** board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"**)**

***BACKTRACKING***

**LOGIC C#**

public bool Exist(char[][] board, string word) {

int rows = board.Length;

int columns = board[0].Length;

var isVisited = new bool[rows, columns];

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < columns; j++)

{

if(word[0] == board[i][j] && searchWord(i, j, board, word, isVisited, 0))

{

return true;

}

}

}

return false;

}

private bool searchWord(int i, int j,char[][] board,string word,bool[,] visited,int index)

{

if (index == word.Length)

return true;

if (i < 0 || j < 0 || i >= board.Length || j >= board[0].Length|| visited[i, j] || board[i][j] != word[index])

return false;

visited[i, j] = true;

//4 directions top bottom left, right

if(searchWord(i+1, j ,board,word, visited, index +1) ||

searchWord(i-1 , j, board, word, visited, index + 1) ||

searchWord(i , j+1, board, word, visited, index + 1)||

searchWord(i , j-1, board, word, visited, index + 1)

)

{

return true;

}

visited[i, j] = false;

return false;

}

**LOGIC JS**

**TODO**

**44. Longest Subsequent sequence ( Input:** nums = [100,4,200,1,3,2]

**Output:** 4**)**

***Arrays***

**LOGIC C#**

public int LongestConsecutive(int[] nums) {

var set = new HashSet<int>();

int n = nums.Length;

int longestSequence = 0;

foreach(int num in nums)

{

set.Add(num);

}

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

{

int currentNum = nums[i];

int sequenceLength = 1;

if (!set.Contains(currentNum - 1))

{

while (set.Contains(currentNum + 1))

{

currentNum += 1;

sequenceLength += 1;

}

longestSequence = Math.Max(longestSequence, sequenceLength);

}

}

return longestSequence;

}

**LOGIC JS**

var longestConsecutive = function(nums) {

let set = new Set();

let longestSequence =0;

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

set.add(nums[i]);

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

let currentNum = nums[i];

let currentLength = 1;

if(!set.has(currentNum-1)){

while(set.has(currentNum +1)){

currentNum +=1;

currentLength +=1;

}

longestSequence = Math.max(longestSequence,currentLength) ;

}

}

return longestSequence;

};

**45.Letter case permutations(Input:** s = "a1b2"

**Output:** ["a1b2","a1B2","A1b2","A1B2"]**)**

**LOGIC C#**

public IList<string> LetterCasePermutation(string s) {

List<string> result = new List<string>();

runDFS(s.ToCharArray(), 0, result);

return result;

}

private static void runDFS(char[] arr, int start, List<string> result)

{

result.Add(new string(arr));

for(int i = start;i<arr.Length; i++)

{

if (char.IsNumber(arr[i]))

continue;

var next = arr.ToArray();

var current = arr[i];

if (char.IsUpper(current))

{

next[i] = char.ToLower(current);

}

else

{

next[i] = char.ToUpper(current);

}

runDFS(next, i + 1, result);

}

}

**LOGIC JS**

var letterCasePermutation = function(s) {

let result= [];

runDFS(s.split(''), 0, result);

return result;

};

function runDFS(arr, start, result) {

result.push(arr.join(''));

for(let i = start;i<arr.length; i++)

{

var next = Array.from(arr);

var current = arr[i];

if(!isNaN(current)){

continue;

}

if (current == current.toUpperCase())

{

next[i] = current.toLowerCase();

}

else

{

next[i] = current.toUpperCase();

}

runDFS(next, i + 1, result);

}

}

**46.SUBSETS( Input:** nums = [1,2,3]

**Output:** [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]**)**

***BACKTRACKING***

**LOGIC C#**

public IList<IList<int>> Subsets(int[] nums) {

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

if (nums == null || nums.Length == 0)

return subsets;

generateSubsets(0, nums, new List<int>(), subsets);

return subsets;

}

private static void generateSubsets(int start, int[] nums, List<int> list, List<IList<int>> subsets)

{

subsets.Add(new List<int>(list));

for (int i = start; i < nums.Length; i++)

{

list.Add(nums[i]);

generateSubsets(i + 1, nums, list, subsets);

list.RemoveAt(list.Count - 1);

}

}

**LOGIC JS**

var subsets = function(nums) {

let subset = [];

let list = []

if(nums.length ==0 || nums == null )

return subset;

generateSubset(0,list,nums,subset);

return subset;

};

function generateSubset(start,list,nums,subset){

subset.push(list.slice());

for(let i = start; i< nums.length;i++){

list.push(nums[i]);

generateSubset(i+1,list,nums,subset);

list.pop();

}

}

**47. SubSets With Duplicate( Input:** nums = [1,2,2]

**Output:** [[],[1],[1,2],[1,2,2],[2],[2,2]]**)**

***BACKTRACKING***

**LOGIC C#**

public IList<IList<int>> SubsetsWithDup(int[] nums)

{

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

Array.Sort(nums);

generateSubsets(0, nums, new List<int>(), subsets);

return subsets;

}

private void generateSubsets(int start, int[] nums, List<int> list, List<IList<int>> subsets)

{

subsets.Add(new List<int>(list));

for (int i = start; i < nums.Length; i++)

{

if (i > start && nums[i - 1] == nums[i])

continue;

list.Add(nums[i]);

generateSubsets(i + 1, nums, list, subsets);

list.RemoveAt(list.Count - 1);

}

}

**LOGIC JS**

var subsetsWithDup = function(nums) {

let subset = [];

let list = []

if(nums.length ==0 || nums == null )

return subset;

generateSubset(0,list,nums.sort(),subset);

return subset;

};

function generateSubset(start,list,nums,subset){

subset.push(list.slice());

for(let i = start; i< nums.length;i++){

if(i>start && nums[i-1] == nums[i])

continue;

list.push(nums[i]);

generateSubset(i+1,list,nums,subset);

list.pop();

}

}

**48. Permutes ( Input:** nums = [1,2,3]

**Output:** [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]**)**

***BACKTRACKING***

**LOGIC C#**

public IList<IList<int>> Permute(int[] nums) {

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

generatePermutations(new List<int>(),nums, permutes);

return permutes;

}

private static void generatePermutations(List<int> list, int[] nums, List<IList<int>> permutes)

{

if(list.Count!= nums.Length)

{

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

{

if (list.Contains(nums[i]))

continue;

list.Add(nums[i]);

generatePermutations(list, nums, permutes);

list.RemoveAt(list.Count - 1);

}

}

else

{

permutes.Add(new List<int>(list));

}

}

**LOGIC JS**

var permute = function(nums) {

let permutes = [];

let list = [];

generatePermutations(list,nums, permutes);

return permutes;

};

function generatePermutations(list,nums, permutes){

if(list.length != nums.length)

{

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

{

if (list.includes(nums[i]))

continue;

list.push(nums[i]);

generatePermutations(list, nums, permutes);

list.pop();

}

}

else

{

permutes.push(list.slice());

}

}

**49. PermutesWith duplicates (Input:** nums = [1,1,2]

**Output:**

[[1,1,2],

[1,2,1],

[2,1,1]]

**)**

***BACKTRACKING***

**LOGIC C#**

public IList<IList<int>> PermuteUnique(int[] nums) {

var n = nums.Length;

var result = new List<IList<int>>();

if (n == 0) return result;

Array.Sort(nums);

generatePermutations(new List<int>(), nums, new bool[n], result);

return result;

}

private void generatePermutations(List<int> list, int[] nums, bool[] isVisited, List<IList<int>> permutes)

{

if (list.Count != nums.Length)

{

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

{

if (isVisited[i]) continue;

if (i > 0 && nums[i - 1] == nums[i] && isVisited[i - 1])

continue;

isVisited[i] = true;

list.Add(nums[i]);

generatePermutations(list, nums, isVisited, permutes);

isVisited[i] = false;

list.RemoveAt(list.Count - 1);

}

}

else

{

permutes.Add(new List<int>(list));

}

}

**LOGIC JS**

var permuteUnique = function(nums) {

var n = nums.length;

let result = [];

if (n == 0)

return result;

let list = [];

let visited = new Array(n);

visited.fill(false);

generatePermutations(list, nums.sort(),visited, result);

return result;

};

function generatePermutations(list, nums,isVisited, permutes){

if (list.length != nums.length)

{

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

{

if (isVisited[i])

continue;

if (i > 0 && nums[i - 1] == nums[i] && isVisited[i - 1])

continue;

isVisited[i] = true;

list.push(nums[i]);

generatePermutations(list, nums, isVisited, permutes);

isVisited[i] = false;

list.pop();

}

}

else

{

permutes.push(list.slice());

}

}

**50. Combinations ( Input:** n = 4, k = 2

**Output:**

[

[2,4],

[3,4],

[2,3],

[1,2],

[1,3],

[1,4],

]

**)**

***BACKTRACKING***

**LOGIC C#**

public IList<IList<int>> Combine(int n, int k) {

var result = new List<IList<int>>();

generateCombinations(1,n,k,new List<int>(), result);

return result;

}

private void generateCombinations(int start, int n, int k,List<int> list, List<IList<int>> result)

{

if (list.Count == k)

result.Add(new List<int>(list));

else

{

for(int i = start; i<=n; i++)

{

list.Add(i);

generateCombinations(i + 1, n, k, list, result);

list.RemoveAt(list.Count - 1);

}

}

}

**LOGIC JS**

var combine = function(n, k) {

let result = [];

let list = [];

generateCombinations(1,n,k,list,result);

return result;

};

function generateCombinations(start,n,k,list,result) {

if (list.length == k)

result.push(list.slice());

else

{

for(let i = start; i<=n; i++)

{

list.push(i);

generateCombinations(i + 1, n, k, list, result);

list.pop();

}

}

}

**51. Combination Sum( Input:** candidates = [2,3,6,7], target = 7

**Output:** [[2,2,3],[7]]**)**

***BACKTRACKING***

**LOGIC C#**

public IList<IList<int>> CombinationSum(int[] candidates, int target) {

var result = new List<IList<int>>();

if (candidates.Length == 0) return result;

generateCombinationSum(0,candidates, target, new List<int>(), result);

return result;

}

private void generateCombinationSum(int start,int[] candidates, int target, IList<int> list, IList<IList<int>> result)

{

if (target == 0)

{

result.Add(new List<int>(list));

}

else if (target > 0)

{

for (int i = start; i < candidates.Length; i++)

{

list.Add(candidates[i]);

generateCombinationSum(i, candidates, target - candidates[i], list, result);

list.RemoveAt(list.Count - 1);

}

}

}

**LOGIC JS**

var combinationSum = function(candidates, target) {

let result = [];

let list = [];

if (candidates.length == 0)

return result;

generateCombinationSum(0,candidates, target, list, result);

return result;

};

function generateCombinationSum(start,candidates, target, list, result){

if(target == 0)

result.push(list.slice());

else if(target> 0){

for(let i =start; i<candidates.length; i++){

list.push(candidates[i]);

generateCombinationSum(i,candidates, target-candidates[i], list, result);

list.pop();

}

}

}

**52.Combination Sum ii(Input:** candidates = [10,1,2,7,6,1,5], target = 8

**Output:**

[

[1,1,6],

[1,2,5],

[1,7],

[2,6]

]

**)**

**LOGIC C#**

public IList<IList<int>> CombinationSum2(int[] candidates, int target) {

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

if (candidates.Length == 0) return result;

Array.Sort(candidates);

generateSum(0, new List<int>(), candidates, target,result);

return result;

}

private void generateSum(int start, List<int> list, int[] candidates, int target,List<IList<int>> result)

{

if (target == 0)

result.Add(new List<int>(list));

else

{

for(int i = start; i<candidates.Length; i++)

{

if (i > start && candidates[i] == candidates[i - 1])

continue;

if(target-candidates[i]<0)

continue;

list.Add(candidates[i]);

generateSum(i + 1, list, candidates, target - candidates[i], result);

list.RemoveAt(list.Count - 1);

}

}

}

**LOGIC JS**

var combinationSum2 = function(candidates, target) {

let result = [];

if (candidates.length == 0)

return result;

let list = [];

generateSum(0, list, candidates.sort(), target,result);

return result;

};

function generateSum(start, list, candidates, target,result){

if (target == 0){

result.push(list.slice());

}

else{

for(let i = start; i<candidates.length; i++)

{

if (i > start && candidates[i] == candidates[i - 1])

continue;

if(target-candidates[i]<0)

continue;

list.push(candidates[i]);

generateSum(i + 1, list, candidates, target - candidates[i], result);

list.pop();

}

}

}

**53.Combination Sum iii( Input:** k = 3, n = 7

**Output:** [[1,2,4]]

**Explanation:**

1 + 2 + 4 = 7

**)**

**LOGIC C#**

public IList<IList<int>> CombinationSum3(int k, int n) {

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

generateCombination(1, k, n,new List<int>(), result);

return result;

}

private void generateCombination(int start, int k, int n,List<int>list, List<IList<int>> result)

{

if (n < 0) return;

if (k == 0 && n ==0)

{

result.Add(new List<int>(list));

}

else

{

for (int i = start; i <= 9; i++)

{

list.Add(i);

generateCombination(i + 1, k - 1, n - i, list, result);

list.RemoveAt(list.Count - 1);

}

}

}

**LOGIC JS**

var combinationSum3 = function(k, n) {

let result = [];

let list = [];

generateCombination(1, k, n,list, result);

return result;

};

function generateCombination(start, k, n,list, result){

if(n<0)

return;

if(k ==0 && n == 0)

{

result.push(list.slice());

}

else{

for(let i = start; i<= 9; i++){

list.push(i);

generateCombination(i+1,k-1,n-i,list,result);

list.pop();

}

}

}

**54. Generate Parenthesis (Input:** n = 3

**Output:** ["((()))","(()())","(())()","()(())","()()()"])

***BACKTRACKING***

**LOGIC C#**

public IList<string> GenerateParenthesis(int n)

{

List<string> result = new List<string>();

GenerateAndCheck("", 0, 0,n,result);

return result;

}

private void GenerateAndCheck(string str, int opened, int closed, int maxLen, List<string> result )

{

if (opened == closed && opened == maxLen)

{

result.Add(str);

return;

}

if (opened < maxLen)

GenerateAndCheck(str + "(", opened + 1, closed,maxLen,result );

if (closed < opened)

GenerateAndCheck(str + ")", opened, closed + 1,maxLen,result);

}

**LOGIC JS**

var generateParenthesis = function(n) {

let result = [];

backtrack(result,'',0,0,n);

return result;

};

function backtrack(result,str,open,close,maxLen){

if(open == close && open == maxLen){

result.push(str);

}

if(open<maxLen)

backtrack(result,str +'(',open+1,close,maxLen);

if(close<open)

backtrack(result,str +')',open,close+1,maxLen);

}

**55.TargetSum ( Input:** nums = [1,1,1,1,1], target = 3

**Output:** 5

**Explanation:** There are 5 ways to assign symbols to make the sum of nums be target 3.

-1 + 1 + 1 + 1 + 1 = 3

+1 - 1 + 1 + 1 + 1 = 3

+1 + 1 - 1 + 1 + 1 = 3

+1 + 1 + 1 - 1 + 1 = 3

+1 + 1 + 1 + 1 - 1 = 3

**)**

***DFS***

**LOGIC C#**

int result = 0;

public int FindTargetSumWays(int[] nums, int target) {

if(nums == null || nums.Length == 0)

return result;

backtrack(nums,0,0,target);

return result;

}

private void backtrack(int[] nums,int start,int currSum, int target){

if(start == nums.Length ){

if(currSum == target)

result++;

return;

}

if(start>= nums.Length)

return;

backtrack(nums, start+1,currSum-nums[start],target);

backtrack(nums, start+1,currSum+nums[start],target);

}

**LOGIC JS**

**TODO**

**56. PALINDROME PARTITIONING (Input:** s = "aab"

**Output:** [["a","a","b"],["aa","b"]]

**)**

***BACKTRACKING***

**LOGIC C#**

public IList<IList<string>> Partition(string s) {

IList<IList<string>> result = new List<IList<string>>();

Part(0,s, result, new List<string>());

return result;

}

public bool IsPali(string s, int left, int right)

{

while (left < right)

{

if (s[left++] != s[right--]) return false;

}

return true;

}

public void Part( int start,string s, IList<IList<string>> result, List<string> list)

{

if (start == s.Length)

{

result.Add(new List<string>(list));

return;

}

for (int i = start; i < s.Length; i++)

{

if (IsPali(s, start, i))

{

list.Add(s.Substring(start, i - start + 1));

Part(i + 1,s, result, list );

list.RemoveAt(list.Count - 1);

}

}

}

**LOGIC JS**

var partition = function(s) {

let result = [];

Part(0,s, result, []);

return result;

};

function Part(start,str,result,list){

if(start == str.length)

result.push(list.slice());

for(let i = start; i<str.length; i++){

if(isPal(str,start,i)){

list.push(str.substr(start, i-start+1));

Part(i+1,str,result,list);

list.pop();

}

}

}

function isPal(str,left,right){

while(left<right){

if(str[left++] != str[right--]){

//left++;

//right--;

return false;

}

}

return true;

}

**57.phone number combination ( Input:** digits = "23"

**Output:** ["ad","ae","af","bd","be","bf","cd","ce","cf"])

***BACKTRACKING***

**LOGIC C#**

private Dictionary<char, string> getMapWithNumberAndStringMapping()

{

Dictionary<char, string> dic = new Dictionary<char, string>();

dic.Add('1', "");

dic.Add('2', "abc");

dic.Add('3', "def");

dic.Add('4', "ghi");

dic.Add('5', "jkl");

dic.Add('6', "mno");

dic.Add('7', "pqrs");

dic.Add('8', "tuv");

dic.Add('9', "wxyz");

return dic;

}

public IList<string> LetterCombinations(string digits)

{

List<string> result = new List<string>();

If(digits == null || digits.Length == 0)

Return result;

letterCombinations(0, digits, "", result, phoneNumberData);

return result;

}

private void letterCombinations(int start, string digits,string data, List<string> result, Dictionary<char, string> phoneNumberdata){

if(start == digits.Length)

{

Result.Add(data)

return;

}

char number = digits[start];

string letters = phoneNumberdata[number];

for(int i = 0;i< letters.Length; i++)

{

letterCombinations(start + 1, digits, data + letters[i], result, phoneNumberdata);

}

}

**LOGIC JS**

var letterCombinations = function(digits) {

let length = digits.length;

if (!length) {

return [];

};

let result = [];

let digitMap =

{'2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl', '6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'};

letterComb(0, digits, "", result, digitMap);

return result;

};

function letterComb(start, digits, data, result, digitMap){

if(start == digits.length)

{

result.push(data);

return;

}

let letters = digitMap[digits[start]];

for(let i = 0;i< letters.length; i++)

{

letterComb(start + 1, digits, data + letters[i], result, digitMap);

}

}

**58. Generalized Abbreviation**

Input:

word = "word",

Output:

["word", "1ord", "w1rd", "wo1d", "wor1", "2rd", "w2d", "wo2", "1o1d", "1or1", "w1r1", "1o2", "2r1", "3d", "w3", "4"]

***BACKTRACKING***

**LOGIC C#**

public List<string> generateAbbreviations(string word)

{

List<string> result = new List<string>();

generateAbbreviationHelper("", word, result,0,0);

return result;

}

public void generateAbbreviationHelper(string data,string word, List<string>result,int count, int position)

{

if(position == word.Length)

{

if(count == 0)

{

result.Add(data);

}

else

{

data += count;

result.Add(data);

}

return;

}

if (count > 0)

generateAbbreviationHelper(data+count+ word[position], word, result,0,position+1);

else

generateAbbreviationHelper( data + word[position], word, result, 0, position + 1);

//No Condition

generateAbbreviationHelper( data , word, result, count+1, position + 1);

}

**LOGIC JS**

let generateAbbreviations = function(word){

let result = [];

generateAbbreviationHelper("", word, result,0,0);

return result;

}

function generateAbbreviationHelper(data, word, result,count,position){

if(position == word.length)

{

if(count == 0)

{

result.push(data);

}

else

{

data += count;

result.push(data);

}

return;

}

if (count > 0)

generateAbbreviationHelper(data+count+ word[position], word, result,0,position+1);

else

generateAbbreviationHelper( data + word[position], word, result, 0, position + 1);

generateAbbreviationHelper( data , word, result, count+1, position + 1);

}

console.log(generateAbbreviations("word"));

**59. House Robber ( Input:** nums = [1,2,3,1]

**Output:** 4**)**

***DYNAMIC PROGRAMMING***

**LOGIC C#**

public int Rob(int[] nums) {

int[] dp = new int[nums.Length + 1];

dp[0] = 0;

dp[1] = nums[0];

for(int i =1; i<nums.Length; i++)

{

dp[i + 1] = Math.Max(dp[i], dp[i - 1] + nums[i]);

}

return dp[nums.Length];

}

**LOGIC JS**

var rob = function(nums) {

let dp = new Array(nums.length+1);

dp[0] = 0;

dp[1] = nums[0];

for(let i = 1; i<nums.length; i++)

dp[i+1] = Math.max(dp[i],dp[i-1]+nums[i]);

return dp[nums.length];

};

**60.HOUSE ROBBER 2 (** 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**.

**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.

**)**

***DYNAMIC PROGRAMMING***

**LOGIC C#**

public int Rob(int[] nums)

{

int len = nums.Length;

if (len == 0)

return 0;

if (len == 1)

return nums[0];

return Math.Max(Robbed(nums, 0, len - 2), Robbed(nums, 1, len - 1));

}

public int Robbed(int[] nums,int start,int end)

{

if (start == end)

return nums[start];

int[] dp = new int[nums.Length ];

dp[start] = nums[start];

dp[start+1] = Math.Max(nums[start], nums[start + 1]);

for (int i = start + 2; i <= end; i++)

dp[i] = Math.Max(dp[i - 1], dp[i - 2]+ nums[i]);

return dp[end];

}

**LOGIC JS**

var rob = function(nums) {

if(nums.length == 0)

return 0;

if(nums.length == 1)

return nums[0];

return Math.max(robbed(nums,0,nums.length-2),robbed(nums,1,nums.length-1));

};

function robbed(nums,start,end){

if(start == end)

return nums[start];

let dp = new Array(nums.length);

dp[start] = nums[start];

dp[start+1] = Math.max(nums[start],nums[start+1]);

for(let i = start+2; i<=end; i++)

dp[i] = Math.max(dp[i-1], dp[i-2] +nums[i]);

return dp[end];

}

**61.Coin Change(Input:** coins = [1,2,5], amount = 11

**Output:** 3

**Explanation:** 11 = 5 + 5 + 1**)**

**D*ynamic Programming***

**LOGIC C#**

public int CoinChange(int[] coins, int amount) {

var dp = Recurse(coins, amount, 0);

return dp[amount] <= amount ? dp[amount] : -1;

}

public int[] Recurse(int[] coins, int amount, int index)

{

if(coins == null || coins.Length <= index)

{

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

Array.Fill(dp, amount + 1);

dp[0] = 0;

return dp;

}

else {

var dp = Recurse(coins, amount, index+1);

int coin = coins[index];

for(int i = coin; i < dp.Length; i++)

{

dp[i] = Math.Min(dp[i], 1 + dp[i - coin]);

}

return dp;

}

}

**LOGIC JS**

var coinChange = function(coins, amount) {

var dp = recurse(coins,amount,0);

return dp[amount]<= amount ? dp[amount] :-1;

};

function recurse(coins,amount,Index){

if(coins == null || coins.length<=Index){

var dp = new Array(amount +1);

dp.fill(amount+1);

dp[0] = 0;

return dp;

}

else

{

var dp = recurse(coins,amount,Index+1);

let coin = coins[Index];

for(let i = coin; i<dp.length; i++){

dp[i] = Math.min(dp[i],1+dp[i-coin]);

}

return dp;

}

}

**62. Maximum Product Subarray**

**Kadens algorithm**

**Input:** nums = [2,3,-2,4]

**Output:** 6

**Explanation:** [2,3] has the largest product 6.

**LOGIC C#**

public int MaxProduct(int[] nums) {

int max = nums[0];

int min = nums[0];

int ans = nums[0];

for (int i = 1; i < nums.Length; i++)

{

if (nums[i] < 0)

{

int temp = max;

max = min;

min = temp;

}

max = Math.Max(nums[i], max \* nums[i]);

min = Math.Min(nums[i], min \* nums[i]);

ans = Math.Max(ans, max);

}

return ans;

}

**LOGIC JS**

var maxProduct = function(nums) {

let min = nums[0];

let max = nums[0];

let ans = nums[0];

for(let i = 1; i<nums.length; i++){

if(nums[i]<0){

let temp = max;

max = min;

min = temp;

}

max = Math.max(nums[i],max\*nums[i]);

min = Math.min(nums[i],min\*nums[i]);

ans = Math.max(ans,max);

}

return ans;

};

**63. Longest Increasing Subsequence**

***Dynamic Programming***

**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

**LOGIC C#**

public int LengthOfLIS(int[] nums)

{

int n = nums.Length;

int len = 0;

if (n > 0)

{

len = 1;

int[] dp = new int[n];

dp[0] = nums[0];

for (int i = 1; i < n; i++)

{

if (nums[i] < dp[0])

dp[0] = nums[i];

else if (nums[i] > dp[len - 1])

dp[len++] = nums[i];

else

dp[BinarySearch(dp, -1, len - 1, nums[i])] = nums[i];

}

}

return len;

}

public int BinarySearch(int[] a, int left, int right, int element)

{

while (right - left > 1)

{

int middle = (right + left) / 2;

if (a[middle] >= element)

right = middle;

else

left = middle;

}

return right;

}

**LOGIC JS**

var lengthOfLIS = function(nums) {

let n = nums.length;

let len = 0;

if (n > 0)

{

len = 1;

let dp = new Array(n);

dp[0] = nums[0];

for (let i = 1; i < n; i++)

{

if (nums[i] < dp[0])

dp[0] = nums[i];

else if (nums[i] > dp[len - 1])

dp[len++] = nums[i];

else

dp[BinarySearch(dp, -1, len - 1, nums[i])] = nums[i];

}

}

return len;

};

function BinarySearch(a, left, right, element){

while (right - left > 1)

{

let middle = Math.floor((left + right) / 2);

if (a[middle] >= element)

right = middle;

else

left = middle;

}

return right;

}

**64.LongestPalindromic Substring**

**Input:** s = "babad"

**Output:** "bab"

**Explanation:** "aba" is also a valid answer.

***Two Pointer***

**LOGIC C#**

public string LongestPalindrome(string s)

{

int maxLength = 0, startIndex = 0;

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

{

int start = i, end = i;

while (end < s.Length - 1 && s[start] == s[end + 1])

end++;

while (end < s.Length - 1 && start > 0 && s[start - 1] == s[end + 1])

{

start--;

end++;

}

if (maxLength < end - start + 1)

{

maxLength = end - start + 1;

startIndex = start;

}

}

return s.Substring(startIndex, maxLength);

}

**LOGIC JS**

var longestPalindrome = function(s) {

let maxLength = 0, startIndex = 0;

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

{

let start = i, end = i;

while (end < s.length - 1 && s[start] == s[end + 1])

end++;

while (end < s.length - 1 && start > 0 && s[start - 1] == s[end + 1])

{

start--;

end++;

}

if (maxLength < end - start + 1)

{

maxLength = end - start + 1;

startIndex = start;

}

}

return s.substr(startIndex, maxLength);

};

**65. WordBreak**

**Input:** s = "leetcode", wordDict = ["leet","code"]

**Output:** true

**Explanation:** Return true because "leetcode" can be segmented as "leet code".

**LOGIC C#**

public bool WordBreak(string s, IList<string> wordDict)

{

var dict = new Dictionary<string, bool>();

return WB(s, wordDict, dict);

}

bool WB(string s, IList<string> wordDict, IDictionary<string, bool> dict)

{

if (dict.ContainsKey(s))

{

return dict[s];

}

foreach (var word in wordDict)

{

if (s == word || s.StartsWith(word) && WB(s.Substring(word.Length, s.Length - word.Length), wordDict, dict))

{

dict[s] = true;

return true;

}

}

dict[s] = false;

return false;

}

**LOGIC JS**

var wordBreak = function(s, wordDict) {

let length = s.length;

let mem = new Array(length).fill(true);

return dfs(0);

function dfs(idx) {

if(idx === length) {

return true;

}

for(let i = idx + 1; i <= length; i++) {

let word = s.substring(i, idx);

if(wordDict.includes(word) && mem[idx]){

if(dfs(i)){

return true;

}

}

}

mem[idx] = false;

return false;

}

};

**66. Combination Sum IV**

**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.

***Dynamic Programming***

**LOGIC C#**

public int CombinationSum4(int[] nums, int target)

{

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

Array.Fill(dp, -1);

return Solve(nums, dp, target);

}

public int Solve(int[] nums, int[] dp, int sum)

{

if (sum == 0)

return 1;

if (sum < 0)

return int.MaxValue;

if (dp[sum] != -1)

return dp[sum];

int ans = 0;

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

{

int res = Solve(nums, dp, sum - nums[i]);

if (res != int.MaxValue)

ans += res;

}

dp[sum] = ans;

return ans;

}

**LOGIC JS**

var combinationSum4 = function(nums, target) {

let dp = new Array(target + 1).fill(-1);

return Solve(nums, dp, target);

};

function Solve(nums, dp,sum)

{

if (sum == 0)

return 1;

if (sum < 0)

return Number.MAX\_SAFE\_INTEGER;

if (dp[sum] != -1)

return dp[sum];

let ans = 0;

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

{

let res = Solve(nums, dp, sum - nums[i]);

if (res != Number.MAX\_SAFE\_INTEGER)

ans += res;

}

dp[sum] = ans;

return ans;

}

**67. Decode Ways**

**Input:** s = "12"

**Output:** 2

**Explanation:** "12" could be decoded as "AB" (1 2) or "L" (12).

**LOGIC C#**

public int NumDecodings(string s)

{

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

dp[0] = 1;

dp[1] = s[0] == '0' ? 0 : 1;

for(int i = 2; i < dp.Length; i++)

{

int oneDigit = Convert.ToInt32(s.Substring(i - 1, 1));

int twoDigit = Convert.ToInt32(s.Substring(i - 2, 2));

if (oneDigit > 0)

{

dp[i] += dp[i - 1];

}

if(twoDigit>=10 && twoDigit <= 26)

{

dp[i] += dp[i - 2];

}

}

return dp[s.Length];

}

**LOGIC JS**

var numDecodings = function(s) {

let dp = new Array(s.length + 1).fill(0);

dp[0] = 1;

dp[1] = s[0] == '0' ? 0 : 1;

for(let i = 2; i < dp.length; i++)

{

let oneDigit =parseInt( s.substr(i - 1, 1));

let twoDigit =parseInt(s.substr(i - 2, 2));

if (oneDigit > 0)

{

dp[i] += dp[i - 1];

}

if(twoDigit>=10 && twoDigit <= 26)

{

dp[i] += dp[i - 2];

}

}

return dp[s.length];

};

**68.unique paths**

**Input:** m = 3, n = 2

**Output:** 3

**Explanation:** From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:

1. Right -> Down -> Down

2. Down -> Down -> Right

3. Down -> Right -> Down

**LOGIC C#**

public int UniquePaths(int m, int n) {

int[][] dp = new int[m][];

// C# thing, as we cannot define int[][] dp = new int[m][n];

for (int i = 0; i < m; i++)

{

dp[i] = new int[n];

}

for (int i = 0; i<dp.Length; i++)

{

dp[i][0] = 1;

}

for (int j = 0; j < dp[0].Length; j++)

{

dp[0][j] = 1;

}

for(int i = 1; i<dp.Length; i++)

{

for(int j = 1; j<dp[0].Length; j++)

{

dp[i][j] = dp[i - 1][j] + dp[i][j - 1];

}

}

return dp[m - 1][n - 1];

}

**LOGIC JS**

var uniquePaths = function(m, n) {

let dp = new Array(m).fill(new Array(n));

for (let i = 0; i<dp.length; i++)

{

dp[i][0] = 1;

}

for (let j = 0; j < dp[0].length; j++)

{

dp[0][j] = 1;

}

for(let i = 1; i<dp.length; i++)

{

for(let j = 1; j<dp[0].length; j++)

{

dp[i][j] = dp[i - 1][j] + dp[i][j - 1];

}

}

return dp[m - 1][n - 1];

};

**69. JUMP GAME**

**Input:** nums = [2,3,1,1,4]

**Output:** true

**Explanation:** Jump 1 step from index 0 to 1, then 3 steps to the last index.

***ITERATIVE***

**LOGIC C#**

public bool CanJump(int[] nums) {

int maxLength = 0;

for (int i = 0; i < nums.Length - 1; i++)

{

if (nums[i] == 0 && maxLength == 0)

{

return false;

}

if (nums[i] >= maxLength)

{

maxLength = nums[i];

}

maxLength--;

}

return true;

}

**LOGIC JS**

var canJump = function(nums) {

let maxLength = 0;

for (let i = 0; i < nums.length - 1; i++)

{

if (nums[i] == 0 && maxLength == 0)

{

return false;

}

if (nums[i] >= maxLength)

{

maxLength = nums[i];

}

maxLength--;

}

return true;

};

**70. Palindromic Substrings**

**Input:** s = "abc"

**Output:** 3

**Explanation:** Three palindromic strings: "a", "b", "c".

**LOGIC C#**

public int CountSubstrings(string s)

{

if(String.IsNullOrEmpty(s)) return 0;

int count = 0;

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

{

//odd

count += ExpandAroundCenter(s, i, i);

//even

count += ExpandAroundCenter(s, i, i+1);

}

return count;

}

private int ExpandAroundCenter(string s, int start, int end)

{

int counter = 0;

while (start >= 0 && end < s.Length && s[start] == s[end])

{

counter++;

start--;

end++;

}

return counter;

}

**LOGIC JS**

var countSubstrings = function(s) {

if(!s) return 0;

let count = 0;

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

{

//odd

count += ExpandAroundCenter(s, i, i);

//even

count += ExpandAroundCenter(s, i, i+1);

}

return count;

};

function ExpandAroundCenter( s, start, end)

{

let counter = 0;

while (start >= 0 && end < s.length && s[start] == s[end])

{

counter++;

start--;

end++;

}

return counter;

}

**71. Number of Longest Increasing Subsequence**

**Input:** nums = [1,3,5,4,7]

**Output:** 2

**Explanation:** The two longest increasing subsequences are [1, 3, 4, 7] and [1, 3, 5, 7].

**LOGIC C#**

public int FindNumberOfLIS(int[] nums) {

if (null == nums || nums.Length == 0) return 0;

int[] len = new int[nums.Length]; //Length of the Longest Increasing Subsequence which ends with nums[i].

int[] count = new int[nums.Length]; //Number of the Longest Increasing Subsequence which ends with nums[i].

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

{

len[i] = 1;

count[i] = 1;

}

for (int i = 1; i < nums.Length; i++)

{

for (int j = 0; j < i; j++)

{

if (nums[j] < nums[i])

{

if (len[j] + 1 > len[i])

{

len[i] = len[j] + 1;

count[i] = count[j];

}

else if (len[j] + 1 == len[i])

{

count[i] += count[j];

}

}

}

}

int maxlen = len.Max();

int ans = 0;

for (int i = 0; i < len.Length; i++)

if (len[i] == maxlen)

ans += count[i];

return ans;

}

**LOGIC JS**

var findNumberOfLIS = function(nums) {

if (null == nums || nums.Length == 0) return 0;

let len = new Array(nums.length).fill(1); //Length of the Longest Increasing Subsequence which ends with nums[i].

let count = new Array(nums.length).fill(1); //Number of the Longest Increasing Subsequence which ends with nums[i].

for (let i = 1; i < nums.length; i++)

{

for (let j = 0; j < i; j++)

{

if (nums[j] < nums[i])

{

if (len[j] + 1 > len[i])

{

len[i] = len[j] + 1;

count[i] = count[j];

}

else if (len[j] + 1 == len[i])

{

count[i] += count[j];

}

}

}

}

let maxlen =Math.max(...len);

let ans = 0;

for (let i = 0; i < len.length; i++)

if (len[i] == maxlen)

ans += count[i];

return ans;

};

**72. Partition Equal Sums**

**Input:** nums = [1,5,11,5]

**Output:** true

**Explanation:** The array can be partitioned as [1, 5, 5] and [11].

***Dynamic Programming***

**LOGIC C#**

public bool CanPartition(int[] nums)

{

int sum = 0;

foreach (int i in nums)

sum += i;

if (sum % 2 != 0)

return false;

int len = nums.Length;

int target = sum / 2;

bool[] dp = new bool[target + 1];

dp[0] = true;

for (int i = 0; i < len; i++)

{

for (int j = target; j >= 0; j--)

{

if (j >= nums[i] && dp[j - nums[i]])

dp[j] = true;

}

}

return dp[target];

}

**LOGIC JS**

var canPartition = function(nums) {

let sum = 0;

let len = nums.length;

for(let i = 0; i<len; i++)

sum += nums[i];

if (sum % 2 != 0)

return false;

let target = sum / 2;

let dp = new Array(target + 1).fill(false);

dp[0] = true;

for (let i = 0; i < len; i++)

{

for (let j = target; j >= 0; j--)

{

if (j >= nums[i] && dp[j - nums[i]])

dp[j] = true;

}

}

return dp[target];

};

**74. Buy Sell Stock With Cooldown**

**Input:** prices = [1,2,3,0,2]

**Output:** 3

**Explanation:** transactions = [buy, sell, cooldown, buy, sell]

**LOGIC C#**

public int MaxProfit(int[] prices) {

if (prices.Length < 2) return 0;

int[] hold = new int[prices.Length];

int[] cash = new int[prices.Length];

// First buy. To buy first share we need to substract share's price from our free cash. Since we have no cash yet, our cash balance is negative

hold[0] = -prices[0];

// We don't buy anything and thus have zero cash balance

cash[0] = 0;

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

{

hold[i] = Math.Max(hold[i - 1], i > 1 ? cash[i - 2] - prices[i] : -prices[i]);

cash[i] = Math.Max(cash[i - 1], hold[i - 1] + prices[i]);

}

return cash[prices.Length - 1];

}

**LOGIC JS**

var maxProfit = function(prices) {

if (prices.length < 2)

return 0;

let hold = new Array(prices.length);

let cash = new Array(prices.length);

hold[0] = -prices[0];

cash[0] = 0;

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

{

hold[i] = Math.max(hold[i - 1], i > 1 ? cash[i - 2] - prices[i] : -prices[i]);

cash[i] = Math.max(cash[i - 1], hold[i - 1] + prices[i]);

}

return cash[prices.length - 1];

};

**75. linked List Cycle II**

**Input:** head = [3,2,0,-4], pos = 1

**Output:** tail connects to node index 1

**Explanation:** There is a cycle in the linked list, where tail connects to the second node.

**LOGIC C#**

public ListNode DetectCycle(ListNode head) {

if (head == null)

return null;

ListNode slow = head;

ListNode fast = head;

while(fast != null && fast.next != null) {

slow = slow.next;

fast = fast.next.next;

if (slow == fast) break;

}

if (fast == null || fast.next == null) return null;

slow = head;

while(slow != fast) {

slow = slow.next;

fast = fast.next;

}

return fast;

}

**LOGIC JS**

var detectCycle = function(head) {

if (head == null)

return null;

let slow = head;

let fast = head;

while(fast != null && fast.next != null) {

slow = slow.next;

fast = fast.next.next;

if (slow == fast) break;

}

if (fast == null || fast.next == null)

return null;

slow = head;

while(slow != fast) {

slow = slow.next;

fast = fast.next;

}

return fast;

};

**76. Addn2 numbers**

**Input:** l1 = [2,4,3], l2 = [5,6,4]

**Output:** [7,0,8]

**Explanation:** 342 + 465 = 807.

**LOGIC C#**

public ListNode AddTwoNumbers(ListNode l1, ListNode l2) {

int carry = 0;

ListNode dummy = new ListNode(0);

ListNode pre = dummy;

while (l1 != null || l2 != null || carry == 1)

{

int sum = (l1 == null ? 0 : l1.val) + (l2 == null ? 0 : l2.val) + carry;

carry = sum < 10 ? 0 : 1;

pre.next = new ListNode(sum % 10);

pre = pre.next;

if (l1 != null)

{

l1 = l1.next;

}

if (l2 != null)

{

l2 = l2.next;

}

}

return dummy.next;

}

**LOGIC JS**

var addTwoNumbers = function(l1, l2) {

let carry = 0;

let dummy = new ListNode(0);

let pre = dummy;

while (l1 != null || l2 != null || carry == 1)

{

let sum = (l1 == null ? 0 : l1.val) + (l2 == null ? 0 : l2.val) + carry;

carry = sum < 10 ? 0 : 1;

pre.next = new ListNode(sum % 10);

pre = pre.next;

if (l1 != null)

{

l1 = l1.next;

}

if (l2 != null)

{

l2 = l2.next;

}

}

return dummy.next;

};

**77. Remove Nth Node From End of List**

**Input:** head = [1,2,3,4,5], n = 2

**Output:** [1,2,3,5]

**Logic c#**

public ListNode RemoveNthFromEnd(ListNode head, int n) {

ListNode dummyHead = new ListNode(0);

dummyHead.next = head;

ListNode slow = dummyHead;

ListNode fast = dummyHead;

for(int i = 0; i<=n; i++)

{

fast = fast.next;

}

while(fast!= null)

{

slow = slow.next;

fast = fast.next;

}

slow.next = slow.next.next;// removes the node

return dummyHead.next;

}

**Logic js**

var removeNthFromEnd = function(head, n) {

let dummyHead = new ListNode(0);

dummyHead.next = head;

let slow = dummyHead;

let fast = dummyHead;

for(let i = 0; i<=n; i++)

{

fast = fast.next;

}

while(fast!= null)

{

slow = slow.next;

fast = fast.next;

}

slow.next = slow.next.next;// removes the node

return dummyHead.next;

};

**78. Sort List**

**Input:** head = [4,2,1,3]

**Output:** [1,2,3,4]

**LOGIC C#**

public ListNode SortList(ListNode head) {

if (head == null || head.next == null) return head;

ListNode temp = head;

ListNode slow = head;

ListNode fast = head;

while(fast!= null && fast.next!= null)

{

temp = slow;

slow = slow.next;

fast = fast.next.next;

}

temp.next = null;

ListNode left = SortList(head);

ListNode right = SortList(slow);

return mergedList(left, right);

}

private ListNode mergedList(ListNode l1, ListNode l2)

{

ListNode sortedTemp = new ListNode(0);

ListNode currentNode = sortedTemp;

while(l1 != null && l2!= null)

{

if (l1.val < l2.val)

{

currentNode.next = l1;

l1 = l1.next;

}

else

{

currentNode.next = l2;

l2 = l2.next;

}

currentNode = currentNode.next;

}

if(l1!= null)

{

currentNode.next = l1;

l1 = l1.next;

}

if (l2 != null)

{

currentNode.next = l2;

l2 = l2.next;

}

return sortedTemp.next;

}

**LOGIC JS**

var sortList = function(head) {

if (head == null || head.next == null) return head;

let temp = head;

let slow = head;

let fast = head;

while(fast!= null && fast.next!= null)

{

temp = slow;

slow = slow.next;

fast = fast.next.next;

}

temp.next = null;

let left = sortList(head);

let right = sortList(slow);

return mergedList(left, right);

};

function mergedList(l1, l2){

let sortedTemp = new ListNode(0);

let currentNode = sortedTemp;

while(l1 != null && l2!= null)

{

if (l1.val < l2.val)

{

currentNode.next = l1;

l1 = l1.next;

}

else

{

currentNode.next = l2;

l2 = l2.next;

}

currentNode = currentNode.next;

}

if(l1!= null)

{

currentNode.next = l1;

l1 = l1.next;

}

if (l2 != null)

{

currentNode.next = l2;

l2 = l2.next;

}

return sortedTemp.next;

}

**79.REORDER LIST**

L0 → L1 → … → Ln - 1 → Ln

*Reorder the list to be on the following form:*

L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …

**Input:** head = [1,2,3,4]

**Output:** [1,4,2,3]

**LOGIC C#**

public void ReorderList(ListNode head) {

if (head == null || head.next == null)

return;

//head of 1st half

ListNode l1 = head;

//head of 2nd half

ListNode slow = head;

//tail of 2nd half

ListNode fast = head;

//tail of 1st half

ListNode temp = null;

while (fast != null && fast.next != null)

{

temp = slow;

slow = slow.next;

fast = fast.next.next;

}

temp.next = null; //makes the middle's next to null;

ListNode l2 = reverse(slow);

merge(l1, l2);

}

private void merge(ListNode l1, ListNode l2)

{

while(l1 != null )

{

ListNode l1Next = l1.next;

ListNode l2Next = l2.next;

l1.next = l2;

if (l1Next == null)

break;

l2.next = l1Next;

l1 = l1Next;

l2 = l2Next;

}

}

public ListNode reverse(ListNode head)

{

ListNode prev = null;

ListNode currentNode = head;

while(currentNode != null)

{

ListNode nextNode = currentNode.next;

currentNode.next = prev;

prev = currentNode;

currentNode = nextNode;

}

return prev;

}

**LOGIC JS**

var reorderList = function(head) {

if (head == null || head.next == null)

return;

//head of 1st half

let l1 = head;

//head of 2nd half

let slow = head;

//tail of 2nd half

let fast = head;

//tail of 1st half

let temp = null;

while (fast != null && fast.next != null)

{

temp = slow;

slow = slow.next;

fast = fast.next.next;

}

temp.next = null; //makes the middle's next to null;

let l2 = reverse(slow);

merge(l1, l2);

};

function merge(l1,l2){

while(l1 != null )

{

let l1Next = l1.next;

let l2Next = l2.next;

l1.next = l2;

if (l1Next == null)

break;

l2.next = l1Next;

l1 = l1Next;

l2 = l2Next;

}

}

function reverse(head){

let prev = null;

let currentNode = head;

while(currentNode != null)

{

let nextNode = currentNode.next;

currentNode.next = prev;

prev = currentNode;

currentNode = nextNode;

}

return prev;

}

**80.Clone a Graph**

**TODO**

**81. Pacific-Atlantic Ocean**

**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]]

**LOGIC C#**

public IList<IList<int>> PacificAtlantic(int[][] heights) {

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

if(heights == null || heights.Length == 0)

return res;

int m = heights.Length, n = heights[0].Length;

bool[,] pacific = new bool[m,n];

bool[,] atlantic = new bool[m,n];

for(int row = 0; row < m; row++)

{

DFS(row, 0, heights, pacific, heights[row][0]);

DFS(row, n - 1, heights, atlantic, heights[row][n - 1]);

}

for(int col = 0; col < n; col++)

{

DFS(0 , col, heights, pacific, heights[0][col]);

DFS(m - 1, col, heights, atlantic, heights[m - 1][col]);

}

for(int i = 0; i<m; i++)

{

for(int j = 0; j<n; j++)

{

if(pacific[i, j] && atlantic[i, j])

res.Add(new List<int>(){i,j

});

}

}

return res;

}

private void DFS(int row, int col, int[][] matrix, bool[,] reach, int prev)

{

int m = matrix.Length, n = matrix[0].Length;

if (row < 0 || row >= m || col < 0 || col >= n || reach[row, col] || matrix[row][col] < prev)

return;

reach[row, col] = true;

DFS(row, col + 1, matrix, reach, matrix[row][col]);

DFS(row, col - 1, matrix, reach, matrix[row][col]);

DFS(row + 1, col, matrix, reach, matrix[row][col]);

DFS(row - 1, col, matrix, reach, matrix[row][col]);

}

**LOGIC JS**

let PacificAtlantic = function(heights) {

let result = [];

if (heights.length === 0 || heights === null)

return result;

let m = heights.length, n = heights[0].length;

let atlantic = new Array(heights.length).fill(false).map(row => new Array(heights[0].length).fill(false));

let pacific = new Array(heights.length).fill(false).map(row => new Array(heights[0].length).fill(false));

for (let row = 0; row < m; row++)

{

DFS(row, 0, heights, pacific, heights[row][0]);

DFS(row, n - 1, heights, atlantic, heights[row][n - 1]);

}

for (let col = 0; col < n; col++)

{

DFS(0, col, heights, pacific, heights[0][col]);

DFS(m - 1, col, heights, atlantic, heights[m - 1][col]);

}

for (let i = 0; i < heights.length; i++){

for (let j = 0; j < heights[0].length; j++){

if (atlantic[i][j] && pacific[i][j]){

result.push([i,j])

}

}

}

return result;

}

function DFS( row, col,matrix, reach, prev){

let m = matrix.length, n = matrix[0].length;

if (row < 0 || row >= m || col < 0 || col >= n || matrix[row][col] < prev)

return;

if (reach[row][col] === true)

return

reach[row][col] = true;

DFS( row+1,col,matrix,reach,matrix[row][col])

DFS( row-1,col,matrix,reach,matrix[row][col])

DFS( row,col+1,matrix,reach,matrix[row][col])

DFS( row, col-1,matrix,reach,matrix[row][col])

}

let 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]];

console.log(PacificAtlantic(heights));

**82. Number of ISLANDS**

**Input:** grid = [

["1","1","1","1","0"],

["1","1","0","1","0"],

["1","1","0","0","0"],

["0","0","0","0","0"]

]

**Output:** 1

**LOGIC C#**

public int NumIslands(char[][] grid) {

int count = 0;

for(int i =0; i<grid.Length; i++){

for(int j = 0; j<grid[i].Length; j++){

if(grid[i][j] == '1'){

count+= 1;

DFS(grid, i,j);

}

}

}

return count;

}

public void DFS(char[][] grid, int i , int j){

if(i<0|| i>=grid.Length || j<0 ||j>= grid[i].Length|| grid[i][j]== '0')

return;

grid[i][j] = '0';

DFS(grid, i+1 ,j);

DFS(grid, i-1 ,j);

DFS(grid, i ,j-1);

DFS(grid, i ,j+1);

}

**LOGIC JS**

var numIslands = function(grid) {

let count = 0;

for (let i = 0; i < grid.length; i++)

{

for (let j = 0; j < grid[i].length; j++)

{

if (grid[i][j] == '1')

{

count += 1;

DFS(grid, i, j);

}

}

}

return count;

};

function DFS(grid, i, j)

{

if (i < 0 || i >= grid.length || j < 0 || j >= grid[i].length || grid[i][j] == '0')

return;

grid[i][j] = '0';

DFS(grid, i + 1, j);

DFS(grid, i - 1, j);

DFS(grid, i, j - 1);

DFS(grid, i, j + 1);

}

**83. Graph valid Tree**

**LOGIC C#**

class Graph

{

private int V; // No. of vertices

private List<int>[] adj; // Adjacency List

// Constructor

Graph(int v)

{

V = v;

adj = new List<int>[v];

for (int i = 0; i < v; ++i)

adj[i] = new List<int>();

}

// Function to add an edge

// into the graph

void addEdge(int v, int w)

{

adj[v].Add(w);

adj[w].Add(v);

}

// A recursive function that uses visited[]

// and parent to detect cycle in subgraph

// reachable from vertex v.

bool isCyclicUtil(int v, bool[] visited,

int parent)

{

// Mark the current node as visited

visited[v] = true;

int i;

// Recur for all the vertices

// adjacent to this vertex

foreach (int it in adj[v])

{

i = it;

// If an adjacent is not visited,

// then recur for that adjacent

if (!visited[i])

{

if (isCyclicUtil(i, visited, v))

return true;

}

// If an adjacent is visited and

// not parent of current vertex,

// then there is a cycle.

else if (i != parent)

return true;

}

return false;

}

// Returns true if the graph

// is a tree, else false.

bool isTree()

{

// Mark all the vertices as not visited

// and not part of recursion stack

bool[] visited = new bool[V];

for (int i = 0; i < V; i++)

visited[i] = false;

// The call to isCyclicUtil serves

// multiple purposes. It returns true if

// graph reachable from vertex 0 is cyclcic.

// It also marks all vertices reachable from 0.

if (isCyclicUtil(0, visited, -1))

return false;

// If we find a vertex which is not reachable

// from 0 (not marked by isCyclicUtil(),

// then we return false

for (int u = 0; u < V; u++)

if (!visited[u])

return false;

return true;

}

// Driver Code

public static void Main(String[] args)

{

// Create a graph given

// in the above diagram

Graph g1 = new Graph(5);

g1.addEdge(0, 1);

g1.addEdge(0, 2);

g1.addEdge(0, 3);

g1.addEdge(1, 4);

if (g1.isTree())

Console.WriteLine("Graph is Tree");

else

Console.WriteLine("Graph is not Tree");

Graph g2 = new Graph(5);

g2.addEdge(0,1);

g2.addEdge(1, 2);

g2.addEdge(2, 3);

g2.addEdge(1, 3);

g2.addEdge(1, 4);

if (g2.isTree())

Console.WriteLine("Graph is Tree");

else

Console.WriteLine("Graph is not Tree");

}

}

**LOGIC JS**

class Graph {

constructor(x) {

this.vertex = x;

this.adjList = new Array(x);

for (let i = 0; i < x; i++)

this.adjList[i] = [];

}

addEdge(n, e) {

this.adjList[n].push(e);

this.adjList[e].push(n);

}

isCyclicUtil(v, visited,parent)

{

visited[v] = true;

let i;

for(let it of this.adjList[v].values())

{

i = it;

if (!visited[i])

{

if (this.isCyclicUtil(i, visited, v))

return true;

}

else if (i != parent)

return true;

}

return false;

}

isTree()

{

let visited = new Array(this.vertex).fill(false);

for (let i = 0; i < this.vertex; i++)

visited[i] = false;

if (this.isCyclicUtil(0, visited, -1))

return false;

for (let u = 0; u < this.vertex; u++)

if (!visited[u])

return false;

return true;

}

}

const g = new Graph(5);

g.addEdge(0, 1);

g.addEdge(0,2);

g.addEdge(0,3);

g.addEdge(1,4);

console.log(g.isTree());

**84.CONNECTED COMPONENTS IN GRAPH**

**LOGIC C#**

public class Graph

{

// A user define class to represent a graph.

// A graph is an array of adjacency lists.

// Size of array will be V (number of vertices

// in graph)

int V;

List<int>[] adjListArray;

// constructor

Graph(int V)

{

this.V = V;

// define the size of array as

// number of vertices

adjListArray = new List<int>[V];

// Create a new list for each vertex

// such that adjacent nodes can be stored

for (int i = 0; i < V; i++)

{

adjListArray[i] = new List<int>();

}

}

// Adds an edge to an undirected graph

void addEdge(int src, int dest)

{

// Add an edge from src to dest.

adjListArray[src].Add(dest);

// Since graph is undirected, add an edge from dest

// to src also

adjListArray[dest].Add(src);

}

void DFSUtil(int v, bool[] visited)

{

// Mark the current node as visited and print it

visited[v] = true;

Console.Write(v + " ");

// Recur for all the vertices

// adjacent to this vertex

foreach (int x in adjListArray[v])

{

if (!visited[x]) DFSUtil(x, visited);

}

}

void connectedComponents()

{

int count = 0;

// Mark all the vertices as not visited

bool[] visited = new bool[V];

for (int v = 0; v < V; ++v)

{

if (!visited[v])

{

// print all reachable vertices

// from v

DFSUtil(v, visited);

count++;

Console.WriteLine();

}

}

Console.Write($"The total connected components are :" + count);

}

// Driver code

public static void Main(String[] args)

{

// Create a graph given in the above diagram

Graph g = new Graph(5); // 5 vertices numbered from 0 to 4

g.addEdge(0, 1);

g.addEdge(1, 2);

g.addEdge(3, 4);

Console.WriteLine("Following are connected components");

g.connectedComponents();

}

}

**LOGIC JS**

class Graph

{

constructor(x){

this.vertex = x;

this.adjList = new Array(x);

for(let i =0; i<x; i++)

this.adjList[i] = [];

}

addEdge(n,e)

{

this.adjList[n].push(e);

this.adjList[e].push(n);

}

graphUtil(v,visited){

visited[v] = true;

console.log(v + " ");

for(let i of this.adjList[v].values()){

let n =i;

if(!visited[n]){

this.graphUtil(n,visited);

}

}

}

connectedGraph(){

let visited = [];

let count = 0

for(let j = 0; j<this.vertex; j++){

if(!visited[j])

{

this.graphUtil(j,visited);

count++;

console.log();

}

}

console.log("The total connected components are" +count);

}

}

const g = new Graph(5);

g.addEdge(1, 0);

g.addEdge(2, 3);

g.addEdge(3, 4);

g.connectedGraph();

**85. Reverse linked list ii**

**Input:** head = [1,2,3,4,5], left = 2, right = 4

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

**LOGIC C#**

public ListNode ReverseBetween(ListNode head, int left, int right) {

if(head == null)

return null;

ListNode prev = null;

ListNode currentNode = head;

while(left>1){

prev = currentNode;

currentNode = currentNode.next;

left--;

right--;

}

ListNode connection = prev;

ListNode tail = currentNode;

while(right>0){

ListNode nextNode = currentNode.next;

currentNode.next = prev;

prev = currentNode;

currentNode = nextNode;

right--;

}

if(connection != null){

connection.next = prev;

}

else

head = prev;

tail.next = currentNode;

return head;

}

**LOGIC JS**

var reverseBetween = function(head, left, right) {

if(head == null)

return null;

let prev = null;

let currentNode = head;

while(left>1){

prev = currentNode;

currentNode = currentNode.next;

left--;

right--;

}

let connection = prev;

let tail = currentNode;

while(right>0){

let nextNode = currentNode.next;

currentNode.next = prev;

prev = currentNode;

currentNode = nextNode;

right--;

}

if(connection != null){

connection.next = prev;

}

else

head = prev;

tail.next = currentNode;

return head;

}

**86.Rotate List**

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

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

Explanation : rotate 1 : 5 1 2 3 4

Rotate 2 = 4,5,1,2,3

**LOGIC C#**

public ListNode RotateRight(ListNode head, int k) {

if (head == null)

return head;

int length = 1;

var tail = head;

while (tail.next != null)

{

length++;

tail = tail.next;

}

k = k % length;

if (length == 0)

{

return head;

}

if (k == 0)

{

return head;

}

var current = head;

for( int i = 0; i< length-k-1; i++)

{

current = current.next;

}

ListNode newhead = current.next;

current.next = null;

tail.next = head;

return newhead;

}

**LOGIC JS**

var rotateRight = function(head, k) {

if (head == null)

return head;

let length = 1;

let tail = head;

while (tail.next != null)

{

length++;

tail = tail.next;

}

k = k % length;

if (k == 0)

{

return head;

}

let current = head;

for( let i = 0; i< length-k-1; i++)

{

current = current.next;

}

let newhead = current.next;

current.next = null;

tail.next = head;

return newhead;

};

**87. Swap Nodes in Pairs**

**Input:** head = [1,2,3,4]

**Output:** [2,1,4,3]

**LOGIC C#**

public ListNode SwapPairs(ListNode head) {

ListNode temp = new ListNode(0);

temp.next = head;

ListNode current = temp;

while(current.next != null && current.next.next!= null){

ListNode firstNode = current.next;

ListNode secondNode = current.next.next;

firstNode.next = secondNode.next;

current.next = secondNode;

current.next.next = firstNode;

current = current.next.next;

}

return temp.next;

}

**LOGIC JS**

var swapPairs = function(head) {

let temp = new ListNode(0);

temp.next = head;

let current = temp;

while(current.next != null && current.next.next!= null){

let firstNode = current.next;

let secondNode = current.next.next;

firstNode.next = secondNode.next;

current.next = secondNode;

current.next.next = firstNode;

current = current.next.next;

}

return temp.next;

};

**88. OddEven LL**

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

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

**LOGIC C#**

public ListNode OddEvenList(ListNode head) {

if(head == null)

return null;

ListNode odd = head;

ListNode even = head.next;

ListNode evenHead = even;

while(even != null && even.next != null){

odd.next = even.next;

odd = odd.next;

even.next = odd.next;

even = even.next;

}

odd.next = evenHead;

return head;

}

**LOGIC JS**

var oddEvenList = function(head) {

if(head == null)

return null;

let odd = head;

let even = head.next;

let evenHead = even;

while(even != null && even.next != null){

odd.next = even.next;

odd = odd.next;

even.next = odd.next;

even = even.next;

}

odd.next = evenHead;

return head;

};

**89. Kth Smallest Element in a Sorted Matrix**

**Input:** matrix = [[1,5,9],[10,11,13],[12,13,15]], k = 8

**Output:** 13

**Explanation:** The elements in the matrix are [1,5,9,10,11,12,13,**13**,15], and the 8th smallest number is 13

**LOGIC C#**

public int KthSmallest(int[][] matrix, int k) {

if (matrix == null) return 0;

int n = matrix.Length;

int left = matrix[0][0];

int right = matrix[n- 1][n - 1];

while (left < right)

{

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

int count = 0;

for (int i = 0; i < n; i++)

for (int j = n - 1; j >= 0; j--)

if (matrix[i][j] <= mid)

count++;

if (count < k)

left = mid + 1;

else

right = mid;

}

return left;

}

**LOGIC JS**

var kthSmallest = function(matrix, k) {

if (matrix == null)

return 0;

let n = matrix.length;

let left = matrix[0][0];

let right = matrix[n- 1][n - 1];

while (left < right)

{

let mid = Math.floor(left + (right - left) / 2);

let count = 0;

for (let i = 0; i < n; i++)

for (let j = n - 1; j >= 0; j--)

if (matrix[i][j] <= mid)

count++;

if (count < k)

left = mid + 1;

else

right = mid;

}

return left;

};

**90. Find K Pairs with Smallest Sums**

**Input:** nums1 = [1,7,11], nums2 = [2,4,6], k = 3

**Output:** [[1,2],[1,4],[1,6]]

**Explanation:** The first 3 pairs are returned from the sequence: [1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]

**LOGIC C#**

**//TODO COZ taking more time to execute**

**LOGIC JS**

**//TODO COZ taking more time to execute**

**91. Merge intervals**

**LOGIC C#**

public int[][] Merge(int[][] intervals) {

if(intervals.Length < 2) return intervals;

Array.Sort(intervals, (arr1, arr2) => arr1[0].CompareTo(arr2[0]));

var result = new List<int[]>();

int start\_current = intervals[0][0];

int end\_current = intervals[0][1];

for(int i = 1; i <intervals.Length; i++){

if(intervals[i][0] <= end\_current){

end\_current = Math.Max(end\_current, intervals[i][1]);

continue;

}

result.Add(new[] {start\_current, end\_current});

start\_current = intervals[i][0];

end\_current = Math.Max(end\_current,intervals[i][1]);

}

result.Add(new[] {start\_current, end\_current});

return result.ToArray();

}

**LOGIC JS**

var merge = function(intervals) {

intervals.sort((a, b) => a[0] - b[0])

const res = [intervals[0]]

for (let curr of intervals) {

prev = res[res.length - 1]

if (prev[1] >= curr[0]) {

prev[1] = Math.max(curr[1], prev[1])

} else {

res.push(curr)

}

}

return res

};

**92.  Interval List Intersections**

**Input:** firstList = [[0,2],[5,10],[13,23],[24,25]], secondList = [[1,5],[8,12],[15,24],[25,26]]

**Output:** [[1,2],[5,5],[8,10],[15,23],[24,24],[25,25]]

//1stList : [[0,2],[5,10],[13,23],[24,25] ]

//2ndList : [[1,5],[8,12],[15,24],[25,26]]

//1st condition. [0,2][1,5]

//The start value of 2nd list should be less than end value of 1st list and

//The end val of 2nd list should be greater than start value of 1ss list

//(max(1ststart,2nd start), min(1st end, 2nd end) ==>[1,2];

//Now increase 1stList Pointer as it is less(2) than 2ndList Pointer(5);

//[5,10][1,5]=>[5,5],(5 is lesss than 10 so increase 2ndpointer

**LOGIC C#**

public int[][] IntervalIntersection(int[][] firstList, int[][] secondList) {

if(firstList == null || secondList == null)

return new int[0][];

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

int idx1 = 0, idx2 = 0;

while(idx1 < firstList.Length && idx2 < secondList.Length)

{

int start1 = firstList[idx1][0];

int end1 = firstList[idx1][1];

int start2 = secondList[idx2][0];

int end2 = secondList[idx2][1];

if(start1 <= end2 && start2 <= end1)

{

int start = Math.Max(start1, start2);

int end = Math.Min(end1, end2);

res.Add(new int[]{start, end});

}

if(end1 <= end2)

idx1++;

else

idx2++;

}

return res.ToArray();

}

**LOGIC JS**

var intervalIntersection = function(firstList, secondList) {

if(firstList == null || secondList == null)

return 0;

let res = [];

let idx1 = 0, idx2 = 0;

while(idx1 < firstList.length && idx2 < secondList.length) {

let start1 = firstList[idx1][0];

let end1 = firstList[idx1][1];

let start2 = secondList[idx2][0];

let end2 = secondList[idx2][1];

if(start1 <= end2 && start2 <= end1)

{

let start = Math.max(start1, start2);

let end = Math.min(end1, end2);

res.push([start, end]);

}

if(end1 <= end2)

idx1++;

else

idx2++;

}

return res;

};

**93.  Non-overlapping Intervals**

**Input:** intervals = [[1,2],[2,3],[3,4],[1,3]]

**Output:** 1

**Explanation:** [1,3] can be removed and the rest of the intervals are non-overlapping.

**LOGIC C#**

public int EraseOverlapIntervals(int[][] intervals) {

if (intervals.Length == 0) return 0;

Array.Sort(intervals, (x, y) => x[1].CompareTo(y[1]));

int result = 0;

int end = intervals[0][1];

for (int i = 1; i < intervals.Length; i++)

if (intervals[i][0] >= end)

end = intervals[i][1];

else

result++;

return result;

}

**LOGIC JS**

var eraseOverlapIntervals = function(intervals) {

if (intervals.length == 0)

return 0;

intervals.sort((a, b) => a[1] - b[1]);

let result = 0;

let end = intervals[0][1];

for (let i = 1; i < intervals.length; i++)

if (intervals[i][0] >= end)

end = intervals[i][1];

else

result++;

return result;

};

**94. MEETING ROOMS II**

Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],...] (si < ei), find the minimum number of conference rooms required.)

Input: intervals = [(0,30),(5,10),(15,20)]

Output: 2

Explanation:

We need two meeting rooms

room1: (0,30)

room2: (5,10),(15,20)

**LOGIC C#**

public int minMeetingRooms(Interval[] intervals)

{

int n = intervals.Length;

int[] start = new int[n];

int[] end = new int[n];

for (int k = 0; k < n; k++)

{

start[k] = intervals[k].start;

end[k] = intervals[k].end;

}

Array.Sort(start);

Array.Sort(end);

int i = 0, j = 0, res = 0;

while (i < n)

{

if (start[i] < end[j]) i++;

else if (start[i] > end[j]) j++;

else

{

i++;

j++;

}

res = Math.Max(res, i - j);

}

return res;

}

**LOGIC JS**

minMeetingRooms = function(intervals){

let n = intervals.length;

let start = new Array(n);

let end = new Array(n);

for (let k = 0; k < n; k++)

{

start[k] = intervals[k].start;

end[k] = intervals[k].end;

}

start.sort();

end.sort();

let i = 0, j = 0, res = 0;

while (i < n)

{

if (start[i] < end[j]) i++;

else if (start[i] > end[j]) j++;

else

{

i++;

j++;

}

res = Math.max(res, i - j);

}

return res;

}

**95. Task Scheduler**

**Input:** tasks = ["A","A","A","B","B","B"], n = 2

**Output:** 8

**Explanation:**

A -> B -> idle -> A -> B -> idle -> A -> B

There is at least 2 units of time between any two same tasks.

**LOGIC C#**

public int LeastInterval(char[] tasks, int n) {

int maxFreq = 0, interval = 0, cnt = 0;

int[] freq = new int[26];

foreach (char t in tasks)

{

freq[t - 'A']++;

if (freq[t - 'A'] > maxFreq)

{

maxFreq = freq[t - 'A'];

cnt = 1;

}

else if (freq[t - 'A'] == maxFreq)

cnt++;

}

interval = (maxFreq - 1) \* (n + 1) + cnt;

return interval < tasks.Length ? tasks.Length : interval;

}

**LOGIC JS**

var leastInterval = function(tasks, n) {

let maxFreq = 0, interval = 0, cnt = 0;

let freq = new Array(26).fill(0);

tasks.forEach(function(t) {

freq[t.charCodeAt(0) - 'A'.charCodeAt(0)]++;

if (freq[t.charCodeAt(0) - 'A'.charCodeAt(0)] > maxFreq)

{

maxFreq = freq[t.charCodeAt(0) - 'A'.charCodeAt(0)];

cnt = 1;

}

else if (freq[t.charCodeAt(0) - 'A'.charCodeAt(0)] == maxFreq)

cnt++;

})

interval = (maxFreq - 1) \* (n + 1) + cnt;

return interval < tasks.length ? tasks.length : interval;

};

**96. Minimum Arrows to burst balloons**

**LOGIC C#**

public int FindMinArrowShots(int[][] points) {

if(points == null || points.Length == 0)

return 0;

Array.Sort(points, (a,b) => a[1].CompareTo(b[1]));

int end = points[0][1], cnt = 1;

for(int i = 1; i < points.Length; i++)

{

if(end >= points[i][0])

continue;

cnt++;

end = points[i][1];

}

return cnt;

}

**LOGIC JS**

let findMinArrowShots = function(points) {

if(points == null || points.length == 0)

return 0;

points.sort((a, b) => a[1] - b[1]);

let end = points[0][1], cnt = 1;

for(let i = 1; i < points.length; i++)

{

if(end >= points[i][0])

continue;

cnt++;

end = points[i][1];

}

return cnt;

};

**97. INSERT INTERVAL**

**Input:** intervals = [[1,3],[6,9]], newInterval = [2,5]

**Output:** [[1,5],[6,9]]

**LOGIC C#**

public int[][] Insert(int[][] intervals, int[] newInterval) {

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

int i = 0;

// Step 1 - add all intervals ending before newInterval starts

while (i < intervals.Length && intervals[i][1] < newInterval[0])

result.Add(intervals[i++]);

// Step 2 - update the newInterval by merging with all overlapping intervals

while (i < intervals.Length && intervals[i][0] <= newInterval[1])

{

newInterval[0] = Math.Min(newInterval[0], intervals[i][0]);

newInterval[1] = Math.Max(newInterval[1], intervals[i][1]);

i++;

}

result.Add(newInterval); // add updated interval

// Step 3 - add remaining intervals

while (i < intervals.Length)

result.Add(intervals[i++]);

return result.ToArray();

}

**LOGIC JS**

var insert = function(intervals, newInterval) {

let result = [];

let i = 0;

while (i < intervals.length && intervals[i][1] < newInterval[0])

result.push(intervals[i++]);

while (i < intervals.length && intervals[i][0] <= newInterval[1])

{

newInterval[0] = Math.min(newInterval[0], intervals[i][0]);

newInterval[1] = Math.max(newInterval[1], intervals[i][1]);

i++;

}

result.push(newInterval);

while (i < intervals.length)

result.push(intervals[i++]);

return result;

};

**98. Find Minimum in Rotated Sorted Array**

**Input:** nums = [3,4,5,1,2]

**Output:** 1

**Explanation:** The original array was [1,2,3,4,5] rotated 3 times.

**LOGIC C#**

public int FindMin(int[] nums) {

if (nums.Length == 0)

return -1;

if (nums.Length == 1)

return nums[0];

int left = 0;

int right = nums.Length - 1;

while (left < right)

{

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

if (mid>0 && nums[mid] < nums[mid - 1])

return nums[mid];

else if(nums[left] <= nums[mid] && nums[mid]> nums[right])

left = mid + 1;

else

right = mid - 1;

}

return nums[left];

}

**LOGIC JS**

var findMin = function(nums) {

if (nums.length == 0)

return -1;

if (nums.length == 1)

return nums[0];

let left = 0;

let right = nums.length - 1;

while (left < right)

{

let mid =Math.floor( left + (right - left) / 2);

if (mid>0 && nums[mid] < nums[mid - 1])

return nums[mid];

else if(nums[left] <= nums[mid] && nums[mid]> nums[right])

left = mid + 1;

else

right = mid - 1;

}

return nums[left];

};

**99.  Find Peak Element**

**Input:** nums = [1,2,3,1]

**Output:** 2

**Explanation:** 3 is a peak element and your function should return the index number 2.

**Logic c#**

public int FindPeakElement(int[] nums) {

int left = 0;

int right = nums.Length - 1;

while (left < right)

{

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

if (nums[mid] < nums[mid + 1] )

left = mid + 1;

else

right = mid;

}

return left;

}

**Logic js**

var findPeakElement = function(nums) {

let left = 0;

let right = nums.length - 1;

while (left < right)

{

let mid = Math.floor(left + (right - left) / 2);

if (nums[mid] < nums[mid + 1] )

left = mid + 1;

else

right = mid;

}

return left;

};

**100. Search in Rotated Sorted Array**

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

**Output:** 4

**LOGIC C#**

public int Search(int[] nums, int target) {

if (nums == null || nums.Length == 0)

return -1;

int left = 0;

int right = nums.Length - 1;

//Find smallest index;

while (left < right)

{

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

if (nums[mid] > nums[right])

{

left = mid + 1;

}

else

{

right = mid;

}

}

int start = left;

left = 0;

right = nums.Length - 1;

if(target>=nums[start] && target <= nums[right])

{

left = start;

}

else

{

right = start;

}

while (left <= right)

{

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

if (nums[mid] == target)

return mid;

else if (nums[mid] < target)

left = mid + 1;

else

right = mid - 1;

}

return -1;

}

**LOGIC JS**

var search = function(nums, target) {

if (nums == null || nums.length == 0)

return -1;

let left = 0;

let right = nums.length - 1;

//Find smallest index;

while (left < right)

{

let mid = Math.floor(left + (right - left) / 2);

if (nums[mid] > nums[right])

{

left = mid + 1;

}

else

{

right = mid;

}

}

let start = left;

left = 0;

right = nums.length - 1;

if(target>=nums[start] && target <= nums[right])

{

left = start;

}

else

{

right = start;

}

while (left <= right)

{

let mid = Math.floor(left + (right - left) / 2);

if (nums[mid] == target)

return mid;

else if (nums[mid] < target)

left = mid + 1;

else

right = mid - 1;

}

return -1;

};

**101. Search in Rotated Sorted Array ii**

**Input:** nums = [2,5,6,0,0,1,2], target = 0

**Output:** true

**LOGIC C#**

public bool Search(int[] nums, int target) {

int low = 0, high = nums.Length - 1;

while(low <= high)

{

int mid = low + (high - low) / 2;

if(nums[mid] == target)

return true;

if(nums[low] == nums[mid] && nums[mid] == nums[high])

{

low++;

high--;

}

else if(nums[low] <= nums[mid])

{

if(nums[low] <= target && target <= nums[mid])

high = mid - 1;

else

low = mid + 1;

}

else

{

if(nums[high] >= target && target >= nums[mid])

low = mid + 1;

else

high = mid - 1;

}

}

return false;

}

**LOGIC JS**

var search = function(nums, target) {

let low = 0, high = nums.length - 1;

while(low <= high)

{

let mid = Math.floor(low + (high - low) / 2);

if(nums[mid] == target)

return true;

if(nums[low] == nums[mid] && nums[mid] == nums[high])

{

low++;

high--;

}

else if(nums[low] <= nums[mid])

{

if(nums[low] <= target && target <= nums[mid])

high = mid - 1;

else

low = mid + 1;

}

else

{

if(nums[high] >= target && target >= nums[mid])

low = mid + 1;

else

high = mid - 1;

}

}

return false;

};

**102. Search 2D Matrix**

**Input:** matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3

**Output:** true

**LOGIC C#**

public bool SearchMatrix(int[][] matrix, int target) {

int rows = matrix.Length;

int colums = matrix[0].Length;

int left = 0;

int right = rows \* colums - 1;

while(left<= right)

{

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

int midPointElement = matrix[mid/colums][mid%colums];

if (midPointElement == target)

return true;

if (midPointElement < target)

left = mid + 1;

else

right = mid - 1;

}

return false;

}

**LOGIC JS**

var searchMatrix = function(matrix, target) {

let rows = matrix.length;

let colums = matrix[0].length;

let left = 0;

let right = rows \* colums - 1;

while(left<= right)

{

let mid =Math.floor(left + (right - left) / 2);

let div = Math.floor(mid/colums);

let percentage = Math.floor(mid%colums)

let midPointElement = matrix[div][percentage];

if (midPointElement == target)

return true;

if (midPointElement < target)

left = mid + 1;

else

right = mid - 1;

}

return false;

};

**103. Search2D matrix II**

**Input:** matrix = [[1,4,7,11,15], [2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]], target = 5

**Output:** true

**LOGIC C#**

public bool SearchMatrix(int[][] matrix, int target)

{

int rows = matrix.Length;

int cols = matrix[0].Length;

int left = 0;

int right = rows - 1;

while (right >= 0 && left < cols)

{

if (matrix[right][left] > target)

right--;

else if (matrix[right][left] < target)

left++;

else

return true;

}

return false;

}

**LOGIC JS**

var searchMatrix = function(matrix, target) {

let rows = matrix.length;

let cols = matrix[0].length;

let left = 0;

let right = rows - 1;

while (right >= 0 && left < cols)

{

if (matrix[right][left] > target)

right--;

else if (matrix[right][left] < target)

left++;

else

return true;

}

return false;

};

**104.Find K Closest elements**

**Input:** arr = [1,2,3,4,5], k = 4, x = 3

**Output:** [1,2,3,4]

**Logic c#**

public IList<int> FindClosestElements(int[] arr, int k, int x) {

int left = 0;

int right = arr.Length-k;

while(left< right){

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

if(x-arr[mid]> arr[mid+k]-x){

left = mid+1;

}

else

right = mid;

}

List<int> result = new List<int>();

for(int i = left; i< (left+k); i++)

result.Add(arr[i]);

return result;

}

**Logic js**

var findClosestElements = function(arr, k, x) {

let left = 0;

let right = arr.length-k;

while(left< right){

let mid = Math.floor(left +(right-left)/2);

if(x-arr[mid]> arr[mid+k]-x){

left = mid+1;

}

else

right = mid;

}

let result = [];

for(let i = left; i< (left+k); i++)

result.push(arr[i]);

return result;

};

**105. Minimum Size Subarray Sum**

**Input:** target = 7, nums = [2,3,1,2,4,3]

**Output:** 2

**Explanation:** The subarray [4,3] has the minimal length under the problem constraint.

**LOGIC C#**

public int MinSubArrayLen(int s, int[] nums)

{

int result = int.MaxValue;

int left = 0;

int val\_sum = 0;

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

{

val\_sum += nums[i];

while (val\_sum >= s)

{

result = Math.Min(result, i + 1 - left);

val\_sum -= nums[left];

left++;

}

}

return result != int.MaxValue ? result : 0;

}

**LOGIC JS**

var minSubArrayLen = function(target, nums) {

let result = Number.MAX\_SAFE\_INTEGER;

let left = 0;

let val\_sum = 0;

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

{

val\_sum += nums[i];

while (val\_sum >= target)

{

result = Math.min(result, i + 1 - left);

val\_sum -= nums[left];

left++;

}

}

return result != Number.MAX\_SAFE\_INTEGER ? result : 0;

};

**106. Fruit Into Baskets**

**Input:** fruits = [1,2,1]

**Output:** 3

**Explanation:** We can pick from all 3 trees.

**LOGIC C#**

public int TotalFruit(int[] fruits) {

int lastFruit = -1;

int secondLastFruit = -1;

int lastFruitCount = 0;

int currentMax = 0;

int max = 0;

foreach(int fruit in fruits)

{

if(fruit == lastFruit || fruit == secondLastFruit)//if you see same 1st and second fruit;

{

currentMax += 1;

}

else

{

//differnt fruit, we will remove the 1st fruit count and take last fruit count and next one;

currentMax = lastFruitCount + 1;

}

if (fruit == lastFruit)

lastFruitCount++;

else

lastFruitCount = 1; //new fruit

if (fruit != lastFruit)

{

secondLastFruit = lastFruit;

lastFruit = fruit;

}

max = Math.Max(currentMax, max);

}

return max;

}

**LOGIC JS**

var totalFruit = function(fruits) {

let lastFruit = -1;

let secondLastFruit = -1;

let lastFruitCount = 0;

let currentMax = 0;

let max = 0;

fruits.forEach(function( fruit )

{

if(fruit == lastFruit || fruit == secondLastFruit)

{

currentMax += 1;

}

else

{

currentMax = lastFruitCount + 1;

}

if (fruit == lastFruit)

lastFruitCount++;

else

lastFruitCount = 1;

if (fruit != lastFruit)

{

secondLastFruit = lastFruit;

lastFruit = fruit;

}

max = Math.max(currentMax, max);

});

return max;

};

**107. Permutation in String**

**Input:** s1 = "ab", s2 = "eidbaooo"

**Output:** true

**Explanation:** s2 contains one permutation of s1 ("ba").

**LOGIC C#**

public bool CheckInclusion(string s1, string s2) {

if(s1.Length > s2.Length)

return false;

int len1 = s1.Length, len2 = s2.Length;

int[] freq = new int[26];

int[] freq2 = new int[26];

for(int i = 0; i < len1; i++)

{

freq[s1[i] - 'a']++;

freq2[s2[i] - 'a']++;

}

// fix the size of sliding window as len1

int left = 0 , right = len1 - 1;

while(right < len2)

{

if(IsEqual(freq, freq2))

return true;

right++;

if(right < len2)

freq2[s2[right] - 'a']++;

freq2[s2[left] - 'a']--;

left++;

}

return false;

}

private bool IsEqual(int[] freq1, int[] freq2)

{

for(int i = 0; i < freq1.Length; i++)

{

if(freq1[i] != freq2[i])

return false;

}

return true;

}

**LOGIC JS**

var checkInclusion = function(s1, s2) {

if(s1.length > s2.length)

return false;

let len1 = s1.length, len2 = s2.length;

let freq = new Array(26).fill(0);

let freq2 = new Array(26).fill(0);

for(let i = 0; i < len1; i++)

{

freq[s1[i].charCodeAt(0) - 'a'.charCodeAt(0)]++;

freq2[s2[i].charCodeAt(0) - 'a'.charCodeAt(0)]++;

}

let left = 0 , right = len1 - 1;

while(right < len2)

{

if(IsEqual(freq, freq2))

return true;

right++;

if(right < len2)

freq2[s2[right].charCodeAt(0) - 'a'.charCodeAt(0)]++;

freq2[s2[left].charCodeAt(0) - 'a'.charCodeAt(0)]--;

left++;

}

return false;

};

function IsEqual(freq1,freq2)

{

for(let i = 0; i < freq1.length; i++)

{

if(freq1[i] != freq2[i])

return false;

}

return true;

}

**108. Longest Repeating Character Replacement**

**Input:** s = "ABAB", k = 2

**Output:** 4

**Explanation:** Replace the two 'A's with two 'B's or vice versa.

**LOGIC C#**

public int CharacterReplacement(string s, int k) {

int n = s.Length;

int[] charCounts = new int[26];

int windowStart = 0;

int maxLength = 0;

int maxCount = 0;

for(int windowEnd =0; windowEnd<n; windowEnd++)

{

charCounts[s[windowEnd] - 'A']++;

int currentCharCount = charCounts[s[windowEnd] - 'A'];

maxCount = Math.Max(maxCount, currentCharCount);

while(windowEnd - windowStart - maxCount + 1 > k)

{

charCounts[s[windowStart] - 'A']--;

windowStart++;

}

maxLength = Math.Max(maxLength, windowEnd - windowStart + 1);

}

return maxLength;

}

**LOGIC JS**

var characterReplacement = function(s, k) {

let n = s.length;

let charCounts = new Array(26).fill(0);

let windowStart = 0;

let maxLength = 0;

let maxCount = 0;

for(let windowEnd =0; windowEnd<n; windowEnd++)

{

charCounts[s[windowEnd].charCodeAt(0) - 'A'.charCodeAt(0)]++;

let currentCharCount = charCounts[s[windowEnd].charCodeAt(0) - 'A'.charCodeAt(0)];

maxCount = Math.max(maxCount, currentCharCount);

while(windowEnd - windowStart - maxCount + 1 > k)

{

charCounts[s[windowStart].charCodeAt(0) - 'A'.charCodeAt(0)]--;

windowStart++;

}

maxLength = Math.max(maxLength, windowEnd - windowStart + 1);

}

return maxLength;

};

**109. Longest Substring Without Repeating Characters**

**Input:** s = "abcabcbb"

**Output:** 3

**Explanation:** The answer is "abc", with the length of 3.

**LOGIC C#**

public int LengthOfLongestSubstring(string s) {

int aPointer = 0;

int max = 0;

int bPointer = 0;

HashSet<char> hashSet = new HashSet<char>();

while (bPointer < s.Length)

{

if (!hashSet.Contains(s[bPointer]))

{

hashSet.Add(s[bPointer]);

bPointer++;

max = Math.Max(hashSet.Count, max);

}

else

{

hashSet.Remove(s[aPointer]);

aPointer++;

}

}

return max;

}

**LOGIC JS**

var lengthOfLongestSubstring = function(s) {

let aPointer = 0;

let max = 0;

let bPointer = 0;

let hashSet = new Set();

while (bPointer < s.length)

{

if (!hashSet.has(s[bPointer]))

{

hashSet.add(s[bPointer]);

bPointer++;

max = Math.max(hashSet.size, max);

}

else

{

hashSet.delete(s[aPointer]);

aPointer++;

}

}

return max;

};

**110. Kth Smallest Element in a BST**

**Input:** root = [3,1,4,null,2], k = 1

**Output:** 1

**LOGIC C#**

public int KthSmallest(TreeNode root, int k) {

int[] nums = new int[2];

inorder(root,nums,k);

return nums[1];

}

private void inorder(TreeNode root, int[] nums,int k){

if(root == null)

return;

inorder(root.left,nums,k);

if(++nums[0] == k){

nums[1] = root.val;

return;

}

inorder(root.right,nums,k);

}

**LOGIC JS**

var kthSmallest = function(root, k) {

let array = []

dfs(root)

return array[k - 1]

function dfs(root){

if(!root){

return

}

dfs(root.left)

array.push(root.val)

dfs(root.right)

}

}

**111. K Closest Points to Origin**

**//TODO**

**112. Top K Frequent Elements**

**Input:** nums = [1,1,1,2,2,3], k = 2

**Output:** [1,2]

**LOGIC C#**

public int[] TopKFrequent(int[] nums, int k) {

if (nums == null || nums.Length == 0 || k == 0)

return new int[] { };

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

foreach (var item in nums)

{

if (!dict.ContainsKey(item))

dict.Add(item, 0);

dict[item] += 1;

}

return dict.OrderByDescending(x => x.Value).Select(x => x.Key).Take(k).ToList<int>().ToArray();

}

**LOGIC JS**

var topKFrequent = function(nums, k) {

let map = new Map();

for(let v of nums) {

if(!map.has(v)) {

map.set(v, 1);

} else {

map.set(v, map.get(v) + 1);

}

}

let array = [];

for(let [key, value] of map) {

array.push([key, value]);

}

array.sort(function(a,b) {

return b[1] - a[1];

})

let result = [];

for(let i = 0; i < k; i++) {

result.push(array[i][0]);

}

return result;

};

**113. Sort Characters By Frequency**

**Input:** s = "tree"

**Output:** "eert"

**LOGIC C#**

public string FrequencySort(string s) {

var di = new Dictionary<char, int>();

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

{

if (di.ContainsKey(s[i]))

{

di[s[i]] += 1;

}

else

{

di.Add(s[i], 1);

}

}

var ordered = di.OrderByDescending(x => x.Value).ToList();

var sb = new StringBuilder();

foreach (var val in ordered)

{

sb.Append(val.Key, val.Value);

}

return sb.ToString();

}

**LOGIC JS**

var frequencySort = function(s) {

let length = s.length;

let alphabet = new Array(127).fill(0);

let max = 0;

let hash;

for (let i = 0; i < length; i++) {

alphabet[s.charCodeAt(i)] += 1;

max = Math.max(max, alphabet[s.charCodeAt(i)]);

};

hash = new Array(max + 1).fill('');

for (let i = 0; i < 127; i++) {

let val = alphabet[i];

hash[max - val] += String.fromCharCode(i).repeat(val);

};

return hash.join('');

};

**114.  Kth Largest Element in an Array**

**Input:** nums = [3,2,1,5,6,4], k = 2

**Output:** 5

**LOGIC C#**

var findKthLargest = function(nums, k) {

return QS(nums, 0, nums.length - 1, nums.length - k);

};

function QS(nums,start,end, index){

let pivot = nums[end];

let left = start;

let right = end - 1;

do

{

while (left <= end && nums[left] < pivot)

{

left++;

}

while (right > start && nums[right] >= pivot)

{

right--;

}

if (left < right)

{

let temp = nums[right];

nums[right] = nums[left];

nums[left] = temp;

}

}

while (left < right);

nums[end] = nums[left];

nums[left] = pivot;

if (left == index) return nums[left];

else if (left < index) return QS(nums, left + 1, end, index);

else return QS(nums, start, left - 1, index);

}

**LOGIC JS**

var findKthLargest = function(nums, k) {

return QS(nums, 0, nums.length - 1, nums.length - k);

};

function QS(nums,start,end, index){

let pivot = nums[end];

let left = start;

let right = end - 1;

do

{

while (left <= end && nums[left] < pivot)

{

left++;

}

while (right > start && nums[right] >= pivot)

{

right--;

}

if (left < right)

{

let temp = nums[right];

nums[right] = nums[left];

nums[left] = temp;

}

}

while (left < right);

nums[end] = nums[left];

nums[left] = pivot;

if (left == index) return nums[left];

else if (left < index) return QS(nums, left + 1, end, index);

else return QS(nums, start, left - 1, index);

}

**115. Reorganize String**

**Input:** s = "aab"

**Output:** "aba"

**LOGIC C#**

public string ReorganizeString(string s) {

int length = s.Length;

if (string.IsNullOrEmpty(s))

{

return s;

}

var di = new Dictionary<char, int>();

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

{

if (di.ContainsKey(s[i]))

{

di[s[i]] += 1;

}

else

{

di.Add(s[i], 1);

}

}

var newHash = di.OrderByDescending(x => x.Value);

if (newHash.First().Value > (s.Length + 1) / 2) return "";

char[] answer = new char[length];

int idx = 0;

foreach (var kvp in newHash)

{

int max = kvp.Value;

for (int i = 0; i < max; i++)

{

if (idx < length)

{

answer[idx] = kvp.Key;

idx += 2;

}

else if (answer[0] != kvp.Key)

{

idx = 1;

answer[idx] = kvp.Key;

idx += 2;

}

else return "";

}

}

return new string(answer);

}

**LOGIC JS**

var reorganizeString = function(S) {

let map = {};

for(let i=0; i<S.length; i++){

map[S[i]] = map[S[i]] || 0;

map[S[i]] ++;

}

let keys = Object.keys(map).sort((a,b)=>{

if(map[a]<map[b])

return 1;

return -1;

});

let benchmark = S.length%2?(Math.floor(S.length/2)+1):S.length/2;

if(map[keys[0]] > benchmark)

return "";

let result = [];

let mostFrequency = map[keys[0]];

let index = 0, max = S.length-1;

while(keys.length){

let currKey = keys.shift();

let count = map[currKey];

while(count){

result[index] = currKey;

index = index+2;

if(index>max)

index=1;

count--;

}

}

return result.join("");

};

**116. COURSE SCHEDULE**

**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.

**LOGIC C#**

public bool CanFinish(int numCourses, int[][] prerequisites) {

List<int>[] arr = new List<int>[numCourses];

bool[] visited = new bool[numCourses];

bool[] tmpVisited = new bool[numCourses];

for(int i = 0; i < numCourses; ++i){

arr[i] = new List<int>();

}

for(int i = 0; i < prerequisites.GetLength(0); ++i){

arr[prerequisites[i][1]].Add(prerequisites[i][0]);

}

for(int i = 0; i < numCourses; ++i){

if(!Visit(arr, visited, tmpVisited, i))

return false;

}

return true;

}

private bool Visit(List<int>[] graph, bool[] visited, bool[] tmpVisited, int node){

if(tmpVisited[node])

return false;

if(visited[node])

return true;

tmpVisited[node] = true;

foreach(int n in graph[node]){

if(!Visit(graph, visited, tmpVisited, n))

return false;

}

tmpVisited[node] = false;

visited[node] = true;

return true;

}

**LOGIC JS**

const isCyclic = (graph, visited, index) => {

if (visited[index] == 2) return true;

visited[index] = 2;

const neighbours = graph.get(index) || [];

for(let n of neighbours) {

if (visited[n] != 1) {

if (isCyclic(graph, visited, n)) return true;

}

}

visited[index] = 1;

return false;

}

var canFinish = function (numCourses, prerequisites) {

// 0 unvisited

// 1 processed

// 2 processing

const graph = new Map();

const visited = new Array(numCourses).fill(0);

for (let [course, dependency] of prerequisites)

graph.set(course, (graph.get(course) || new Set()).add(dependency));

for(let i = 0; i < numCourses; i++) {

if (visited[i] == 0) {

if (isCyclic(graph, visited, i)) {

return false;

}

}

}

return true;

};

**117. Course Schedule II**

**Input:** numCourses = 2, prerequisites = [[1,0]]

**Output:** [0,1]

**LOGIC C#**

private int[] \_result;

private int resultIndex = 0;

public int[] FindOrder(int numCourses, int[][] prerequisites) {

\_result = new int[numCourses];

var adjacencyMatrix = new HashSet<int>[numCourses];

for (int i = 0; i < numCourses; i++) {

adjacencyMatrix[i] = new HashSet<int>();

}

foreach (var fromTo in prerequisites) {

var from = fromTo[0];

var to = fromTo[1];

adjacencyMatrix[from].Add(to);

}

var isVisited = new bool[numCourses];

var isAdded = new bool[numCourses];

for (int i = 0; i < numCourses; i++) {

if (isAdded[i]) continue;

var noCyclic = DFS(i, adjacencyMatrix, isVisited, isAdded);

if (!noCyclic) return new int[0];

}

return \_result;

}

private bool DFS(int cur, HashSet<int>[] adjacencyMatrix, bool[] isVisited, bool[] isAdded) {

if (isVisited[cur]) return false;

isVisited[cur] = true;

var nextCourses = adjacencyMatrix[cur];

foreach (var next in nextCourses) {

var oneResult = DFS(next, adjacencyMatrix, isVisited, isAdded);

if (!oneResult) return false;

}

if (!isAdded[cur]) {

\_result[resultIndex] = cur;

resultIndex++;

isAdded[cur] = true;

}

isVisited[cur] = false;

return true;

}

**LOGIC JS**

var findOrder = function(numCourses, prerequisites) {

// build graph: prereq -> next course

let graph = {}

for(let i=0; i < numCourses; i++) {

graph[i] = []

}

for(let [i,j] of prerequisites) {

graph[j].push(i)

}

let cycle = new Set(), seen = new Set(), stack = [];

for(let i=0; i < numCourses; i++) {

if (dfs(i)==='cycle') return []

}

return stack.reverse()

function dfs(i) {

if (cycle.has(i)) return 'cycle'

if (seen.has(i)) return

seen.add(i)

cycle.add(i)

for(let prereq of graph[i]) {

if (dfs(prereq)==='cycle') return 'cycle'

}

cycle.delete(i) // backtracking

stack.push(i)

}

};

**118. Minimum tree Construction**

**//TODO**

**119. Sequence reconstruction**

**//TODO**

**120.Binary tree traversal level order ii**

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

**Output:** [[15,7],[9,20],[3]]

**LOGIC C#**

public IList<IList<int>> LevelOrderBottom(TreeNode root) {

var result = new List<IList<int>>();

if (root == null) return result;

var queue = new Queue<TreeNode>();

queue.Enqueue(root);

while (queue.Count>0)

{

var size = queue.Count;

var oneResult = new List<int>();

for (int s = 0; s < size; s++)

{

var cur = queue.Dequeue();

oneResult.Add(cur.val);

if (cur.left != null)

{

queue.Enqueue(cur.left);

}

if (cur.right != null)

{

queue.Enqueue(cur.right);

}

}

result.Insert(0, oneResult);

}

// result.Reverse();

return result;

}

**LOGIC JS**

var levelOrderBottom = function(root) {

let result = [];

if (root == null)

return result;

let queue = [];

queue.push(root);

while (queue.length>0)

{

let size = queue.length;

let oneResult =[];

for (let s = 0; s < size; s++)

{

let cur = queue.shift();

oneResult.push(cur.val);

if (cur.left != null)

{

queue.push(cur.left);

}

if (cur.right != null)

{

queue.push(cur.right);

}

}

result.push( oneResult);

}

return result.reverse();

};

**121.Binary tree Traversal Level order**

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

**Output:** [[3],[9,20],[15,7]]

**LOGIC C#**

public IList<IList<int>> LevelOrder(TreeNode root) {

var result = new List<IList<int>>();

if (root == null) return result;

var queue = new Queue<TreeNode>();

queue.Enqueue(root);

while (queue.Count>0)

{

var size = queue.Count;

var oneResult = new List<int>();

for (int s = 0; s < size; s++)

{

var cur = queue.Dequeue();

oneResult.Add(cur.val);

if (cur.left != null)

{

queue.Enqueue(cur.left);

}

if (cur.right != null)

{

queue.Enqueue(cur.right);

}

}

result.Add(oneResult);

//result.Insert(0, oneResult);

}

//result.Reverse();

return result;

}

**LOGIC JS**

var levelOrder = function(root) {

let result = [];

if (root == null)

return result;

let queue = [];

queue.push(root);

while (queue.length>0)

{

let size = queue.length;

let oneResult =[];

for (let s = 0; s < size; s++)

{

let cur = queue.shift();

oneResult.push(cur.val);

if (cur.left != null)

{

queue.push(cur.left);

}

if (cur.right != null)

{

queue.push(cur.right);

}

}

result.push( oneResult);

}

return result;

};

**122. Binary Tree Zigzag Level Order Traversal**

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

**Output:** [[3],[20,9],[15,7]]

**LOGIC C#**

public IList<IList<int>> ZigzagLevelOrder(TreeNode root) {

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

if(root == null)

return res;

Queue<TreeNode> queue = new Queue<TreeNode>();

queue.Enqueue(root);

int rowNum = 0;

while(queue.Count > 0)

{

List<int> row = new List<int>();

int count = queue.Count;

for(int i = 0; i < count; i++)

{

TreeNode node = queue.Dequeue();

row.Add(node.val);

if(node.left != null)

queue.Enqueue(node.left);

if(node.right != null)

queue.Enqueue(node.right);

}

if(rowNum % 2 == 1)

row.Reverse();

res.Add(row);

rowNum++;

}

return res;

}

**LOGIC JS**

var zigzagLevelOrder = function(root) {

let res = [];

if (root == null)

return res;

let queue = [];

queue.push(root);

let rowNum = 0;

while (queue.length>0)

{

let size = queue.length;

let oneResult =[];

for (let s = 0; s < size; s++)

{

let cur = queue.shift();

oneResult.push(cur.val);

if (cur.left != null)

{

queue.push(cur.left);

}

if (cur.right != null)

{

queue.push(cur.right);

}

}

if(rowNum % 2 == 1)

oneResult.reverse();

res.push(oneResult);

rowNum++;

}

return res;

};

**123. Populating Next Right Pointers in Each Node**

**Input:** root = [1,2,3,4,5,6,7]

**Output:** [1,#,2,3,#,4,5,6,7,#]

**LOGIC C#**

public Node Connect(Node root) {

if (root == null)

return null;

Node leftMost = root;

while (leftMost.left != null)

{

Node curr = leftMost;

while (curr != null)

{

curr.left.next = curr.right;

if (curr.next != null)

{

curr.right.next = curr.next.left;

}

curr = curr.next;

}

leftMost = leftMost.left;

}

return root;

}

**LOGIC JS**

var connect = function(root) {

if (root == null)

return null;

let leftMost = root;

while (leftMost.left != null)

{

let curr = leftMost;

while (curr != null)

{

curr.left.next = curr.right;

if (curr.next != null)

{

curr.right.next = curr.next.left;

}

curr = curr.next;

}

leftMost = leftMost.left;

}

return root;

};

**124. Populating Next Right Pointers in Each Node II**

**Input:** root = [1,2,3,4,5,null,7]

**Output:** [1,#,2,3,#,4,5,7,#]

**Explanation:** Given the above binary tree (Figure A), your function should populate each next pointer to point to its next right node, just like in Figure B. The serialized output is in level order as connected by the next pointers, with '#' signifying the end of each level.

**LOGIC C#**

**LOGIC JS**

var connect = function(root) {

if(root == null)

return null;

let levelHead = null; // first node of the next level

let prev = null; // last visited node in the next level

let curr = root; // current node in the current level

while(curr != null)

{

while(curr != null)

{

if(curr.left != null)

{

if(prev == null)

{

levelHead = curr.left;

prev = curr.left;

}

else

{

prev.next = curr.left;

prev = curr.left;

}

}

if(curr.right != null)

{

if(prev == null)

{

levelHead = curr.right;

prev = curr.right;

}

else

{

prev.next = curr.right;

prev = curr.right;

}

}

curr = curr.next;

}

// move to the next level

curr = levelHead;

prev = null;

levelHead = null;

}

return root;

}

**125. RIGHT SIDE VIEW OF BINARY TREE**

**Input:** root = [1,2,3,null,5,null,4]

**Output:** [1,3,4]

**LOGIC C#**

public IList<int> RightSideView(TreeNode root) {

List<int> result = new List<int>();

if (root == null)

return result;

var queue = new Queue<TreeNode>();

queue.Enqueue(root);

while (queue.Count > 0)

{

var size = queue.Count;

for (int s = 0; s < size; s++)

{

var cur = queue.Dequeue();

if (s == 0)

{

result.Add(cur.val);

}

if (cur.right != null)

{

queue.Enqueue(cur.right);

}

if (cur.left != null)

{

queue.Enqueue(cur.left);

}

}

}

return result;

}

**LOGIC JS**

var rightSideView = function(root) {

let result = [];

let queue = [];

if(root == null)

return result;

queue.push(root);

while(queue.length>0){

let size = queue.length;

for(let i = 0 ; i<size; i++){

let currentVal = queue.shift();

if(i == 0){

result.push(currentVal.val);

}

if(currentVal.right)

queue.push(currentVal.right);

if(currentVal.left )

queue.push(currentVal.left);

}

}

return result;

};

**126.**

**//TODO**

**127.Path Sum II**

**Input:** root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22

**Output:** [[5,4,11,2],[5,8,4,5]]

**LOGIC C#**

public IList<IList<int>> PathSum(TreeNode root, int targetSum) {

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

findPath(root, targetSum, new List<int>(), result);

return result;

}

public void findPath(TreeNode root, int targetSum, List<int>current, List<IList<int>> paths)

{

if (root == null)

return;

current.Add(root.val);

//reached leafnode

if(root.val == targetSum && root.left==null && root.right== null)

{

paths.Add(current);

}

findPath(root.left, targetSum-root.val, new List<int>(current), paths);

findPath(root.right, targetSum-root.val, new List<int>(current), paths);

}

**LOGIC JS**

class Node

{

constructor(data)

{

this.left = null;

this.right = null;

this.val = data;

}

}

let pathSum = function(root, targetSum) {

let result = [];

findPath(root, targetSum, [], result);

return result;

}

let findPath = function( root, targetSum, current, paths)

{

if (root == null)

return;

current.push(root.val);

if(root.val == targetSum && root.left==null && root.right== null)

{

paths.push(current.slice());

}

findPath(root.left, targetSum-root.val, current, paths);

findPath(root.right, targetSum-root.val,current, paths);

current.pop();

return paths;

}

**128. PATH SUM III**

**Input:** root = [10,5,-3,3,2,null,11,3,-2,null,1], targetSum = 8

**Output:** 3

**LOGIC C#**

int count = 0;

public int PathSum(TreeNode root, int targetSum)

{

List<int> result = new List<int>();

generateSum(root, targetSum, result);

return count;

}

public void generateSum(TreeNode root, int targetSum, List<int> result)

{

if (root == null)

return;

result.Add(root.val);

generateSum(root.left, targetSum, result);

generateSum(root.right, targetSum, result);

int temp = 0;

for(int i =result.Count -1; i>=0; i--)

{

temp += result[i];

if (temp == targetSum)

count++;

}

result.RemoveAt(result.Count - 1);

}

**LOGIC JS**

**//TODO**

**129. LCA OF BINARY TREE**

**Input:** root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

**Output:** 3

**LOGIC C#**

public TreeNode LowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

if (root == null)

{

return root;

}

//// if any one of p or q is root itself. root is itself ancestor to itself

if (root.val == p.val || root.val == q.val)

{

return root;

}

var left = LowestCommonAncestor(root.left, p, q);

var right = LowestCommonAncestor(root.right, p, q);

// This means one of p and q is on right side and other one is on left side

// so ancestor in that case will be root

if (left != null && right != null)

{

return root;

}

// We have already checked for both the condition above

// This means we have found both p and q on left subtree, so return left

if (left != null)

{

return left;

}

// We have already checked for both the condition above

// This means we have found both p and q on right subtree, so return right

if (right != null)

{

return right;

}

// if not found just return null

return null;

}

**LOGIC JS**

let lowestCommonAncestor = function(root, p, q) {

if (root == null)

{

return root;

}

if (root.val == p.val || root.val == q.val)

{

return root;

}

let left = lowestCommonAncestor(root.left, p, q);

let right = lowestCommonAncestor(root.right, p, q);

if (left != null && right != null)

{

return root;

}

if (left != null)

{

return left;

}

if (right != null)

{

return right;

}

return null;

};

**130. Maximum Binary Tree**

**LOGIC C#**

public TreeNode ConstructMaximumBinaryTree(int[] nums) {

return constructMaximumBinaryTree(nums, 0, nums.Length - 1);

}

public TreeNode constructMaximumBinaryTree(int[] nums, int start, int end)

{

if (start > end)

return null;

var maxStartIndex = start;

for(int i =start+1; i<=end; i++)

{

if(nums[i]>nums[maxStartIndex])

{

maxStartIndex = i;

}

}

var node = new TreeNode(nums[maxStartIndex]);

node.left = constructMaximumBinaryTree(nums, start, maxStartIndex - 1);

node.right = constructMaximumBinaryTree(nums, maxStartIndex+1, end);

return node;

}

**LOGIC JS**

function TreeNode(val, left, right) {  
 this.val = (val===undefined ? 0 : val)  
 this.left = (left===undefined ? null : left)  
 this.right = (right===undefined ? null : right)  
 }  
  
  
let constructMaximumBinaryTree = function(nums) {  
 return constructMax(nums, 0, nums.length - 1);  
};  
function constructMax(nums, start, end){  
 if (start > end)  
 return null;  
 let maxStartIndex = start;  
 for(let i =start+1; i<=end; i++)  
 {  
 if(nums[i]>nums[maxStartIndex])  
 {  
 maxStartIndex = i;  
 }  
 }  
 let node = new TreeNode(nums[maxStartIndex]);  
 node.left = constructMax(nums, start, maxStartIndex - 1);  
 node.right = constructMax(nums, maxStartIndex+1, end);  
 return node;  
}  
  
let ***nums*** = [3,2,1,6,0,5];  
***console***.log(constructMaximumBinaryTree(***nums***));