**Date:** 20/11/24

**DSA Practice Problems**

1. **3Sum Closest**

import java.util.Arrays;

class Solution {

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

Arrays.sort(nums);

int close = nums[0] + nums[1] + nums[2];

for (int i = 0; i < nums.length - 2; i++) {

int left = i + 1, right = nums.length - 1;

while (left < right) {

int currentsum = nums[i] + nums[left] + nums[right];

if (currentsum == target) {

return target;

}

if (Math.abs(currentsum - target) < Math.abs(close - target)) {

close = currentsum;

}

if (currentsum < target) {

left++;

} else {

right--;

}

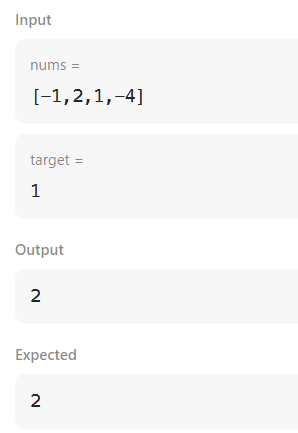
}

}

return close;

}

}



**Time Complexity:** O(n^2)

1. **Jump Game ||**

class Solution {

public int jump(int[] nums) {

int reach = 0;

int count = 0;

int last = 0;

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

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

if (i == last) {

last = reach;

count += 1;

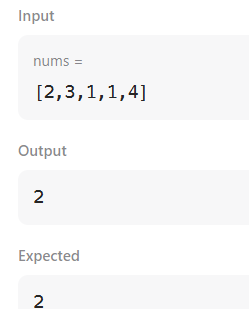
}

}

return count;

}

}



**Time Complexity**: O(n)

1. **Group Anagrams**

class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

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

for (String str : strs) {

char[] chararray = str.toCharArray();

Arrays.sort(chararray);

String sorted = new String(chararray);

map.putIfAbsent(sorted, new ArrayList<>());

map.get(sorted).add(str);

}

return new ArrayList<>(map.values());

}

}



**Time Complexity:** O(n\*k)

1. **Quick sort**

class Solution {

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

quickSort(nums, 0, nums.length - 1);

return nums;

}

static void quickSort(int arr[], int low, int high) {

if (low < high) {

int index = partition(arr, low, high);

quickSort(arr, low, index - 1);

quickSort(arr, index + 1, high);

}

}

static int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i + 1];

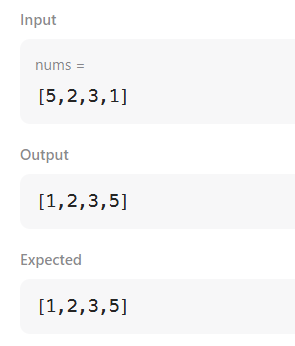
arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

}



**Time Complexity:** O(n log n)

1. **Merge Sort**

class Solution {

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int mid = l + (r - l) / 2;

mergeSort(arr, l, mid);

mergeSort(arr, mid + 1, r);

merge(arr, l, mid, r);

}

}

void merge(int arr[], int l, int mid, int r) {

int n1 = mid - l + 1;

int n2 = r - mid;

int[] L = new int[n1];

int[] R = new int[n2];

System.arraycopy(arr, l, L, 0, n1);

System.arraycopy(arr, mid + 1, R, 0, n2);

int i = 0, j = 0, k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

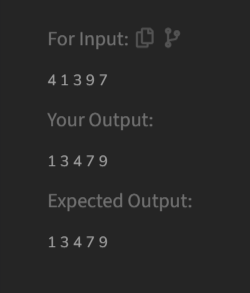
arr[k] = R[j];

j++;

}

k++;

}



**Time Complexity:** O (n log n)

1. **Decode Ways**

class Solution {

public int numDecodings(String s) {

int dp1=1;

int dp2=0;

int dp;

int len=s.length();

for(int i=len-1;i>=0;i--){

if(s.charAt(i)=='0'){

dp=0;

}

else{

dp=dp1;

}

if(i<len-1&& (s.charAt(i)=='1'||s.charAt(i)=='2'&&s.charAt(i+1)<'7')){

dp=dp+dp2;

}

dp2=dp1;

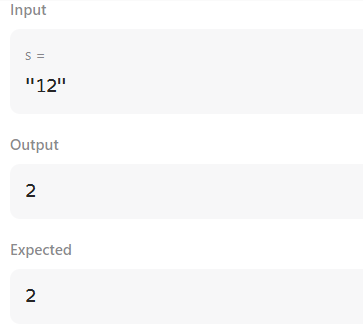
dp1=dp;

}

return dp1;

}

}



**Time Complexity:** O(n)

1. [**Best Time to Buy and Sell Stock II**](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/)

class Solution {

public int maxProfit(int[] prices) {

int sell = 0;

int hold = Integer.MIN\_VALUE;

for (final int price : prices) {

sell = Math.max(sell, hold + price);

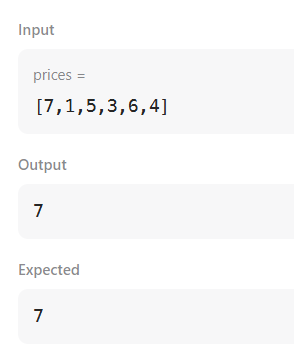
hold = Math.max(hold, sell - price);

}

return sell;

}

}



**Time Complexity:** O(n)

1. **Number Of Islands**

class Solution {

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

if(grid==null || grid.length==0){

return 0;

}

int total=0;

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

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

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

total++;

dfs(grid,i,j);

}

}

}

return total;

}

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

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

return;

}

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

dfs(grid,i+1,j);

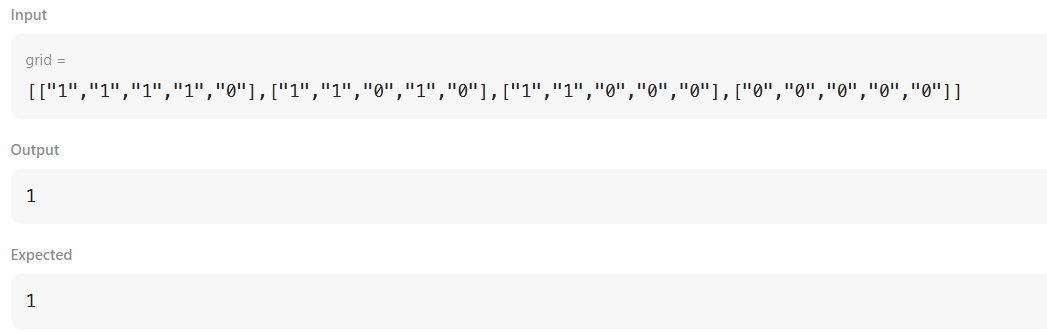
dfs(grid,i-1,j);

dfs(grid,i,j+1);

dfs(grid,i,j-1);

}

}



**Time Complexity:** O (m \* n)

1. **Ternary Search**

class Solution{

static int ternarySearch(int arr[], int N, int K)

{

return ternarysearchmethod(arr,0,N-1,K);

}

static int ternarysearchmethod(int arr[],int l,int r,K){

if(l<=r){

int mid1=l+(r-l)/3;

int mid2=r-(r-l)/3;

if(arr[mid1]==K){

return 1;

}

if(arr[mid2]==K){

return 1;

}

if(K<arr[mid1]){

return ternarysearchmethod(arr,l,mid1-1,K);

}

if(K>arr[mid2]){

return ternarysearchmethod(arr,mid2+1,r,K);

}

else{

return ternarysearchmethod(mid1+1,mid2-1,K);

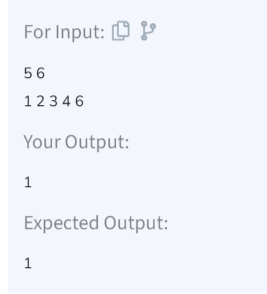
}

}

return -1;

}

}



**Time Complexity:** O (log N)

1. **Interpolation Search**

public class InterpolationSearch {

public static int Searchelement(int arr[],int low,int high,int x){

int index=-1;

if(low<=high && x>=arr[low]&& x<=arr[high]){

int pos=low+(((high-low)/(arr[high]-arr[low]))\*(x-arr[low]));

if(arr[pos]==x){

index=pos;

}

if(arr[pos]<x){

return Searchelement(arr,pos+1,high,x);

}

if(arr[pos]>x){

return Searchelement(arr,low,pos-1,x);

}

}

return index;

}

public static void main(String[] args){

int arr[]={ 10, 12, 13, 16, 18, 19, 20, 21,22, 23, 24, 33, 35, 42, 47 };

int n=arr.length;

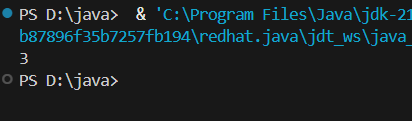
int x=16;

int index=Searchelement(arr,0,n-1,x);

System.out.print(index);

}

}



**Time Complexity:** O(n)