**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--;

}

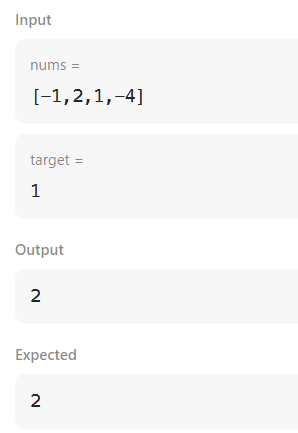
}

}

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;

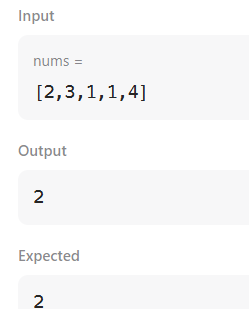
}

}

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];

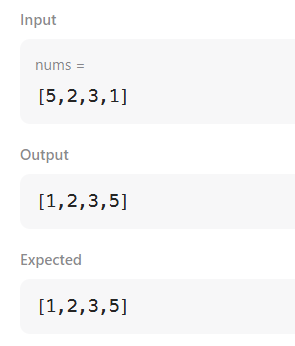
arr[i + 1] = arr[high];

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 {

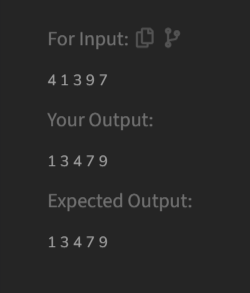
arr[k] = R[j];

j++;

}

k++;

}



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