**ASSIGNMENT3**

1.

function threeSumClosest(num, target) {

num.sort((a, b) => a - b);

let closestSum = Infinity;

for (let i = 0; i < num.length - 2; i++) {

let left = i + 1;

let right = num.length - 1;

while (left < right) {

const sum = num [i] + num [left] + num [right];

if (sum === target) {

return target;

}

if (Math.abs(sum - target) < Math.abs(closestSum - target)) {

closestSum = sum;

}

if (sum > target) {

right--;

} else {

left++;

}

}

}

return closestSum;

}

**// Test case**

const num = [-1, 2, 1, -4];

const target = 1;

console.log(threeSumClosest(nums, target));

**Output:** 2

**2.**

function fourSum(nums, target) {

nums.sort((a, b) => a - b);

const result = [];

for (let i = 0; i < nums.length - 3; i++) {

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

continue;

}

for (let j = i + 1; j < nums.length - 2; j++) {

if (j > i + 1 && nums[j] === nums[j - 1]) {

continue;

}

let left = j + 1;

let right = nums.length - 1;

while (left < right) {

const sum = nums[i] + nums[j] + nums[left] + nums[right];

if (sum === target) {

result.push([nums[i], nums[j], nums[left], nums[right]]);

left++;

right--;

while (left < right && nums[left] === nums[left - 1]) {

left++;

}

while (left < right && nums[right] === nums[right + 1]) {

right--;

}

} else if (sum > target) {

right--;

} else {

left++;

}

}

}

}

return result;

}

**// Test case**

const nums = [1, 0, -1, 0, -2, 2];

const target = 0;

console.log(fourSum(nums, target));

**Output:** [[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]

3.

function nextPermutation(nums) {

let i = nums.length - 2;

while (i >= 0 && nums[i] >= nums[i + 1]) {

i--;

}

if (i >= 0) {

let j = nums.length - 1;

while (j > i && nums[j] <= nums[i]) {

j--;

}

// Swap nums[i] and nums[j]

swap(nums, i, j);

}

// Reverse the subarray nums[i+1:]

reverse(nums, i + 1);

return nums;

}

function swap(nums, i, j) {

const temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

}

function reverse(nums, start) {

let i = start;

let j = nums.length - 1;

while (i < j) {

swap(nums, i, j);

i++;

j--;

}

}

**// Test case**

const nums = [1, 2, 3];

console.log(nextPermutation(nums));

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

**4.**

function searchInsert(nums, target) {

let left = 0;

let right = nums.length - 1;

while (left <= right) {

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

if (nums[mid] === target) {

return mid;

} else if (nums[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return left;

}

**// Test case**

const nums = [1, 3, 5, 6];

const target = 5;

console.log(searchInsert(nums, target));

**Output:** 2

**5.**

function plusOne(digits) {

const n = digits.length;

let carry = 1;

for (let i = n - 1; i >= 0; i--) {

const sum = digits[i] + carry;

digits[i] = sum % 10;

carry = Math.floor(sum / 10);

if (carry === 0) {

// No more carry, early exit

break;

}

}

if (carry === 1) {

digits.unshift(carry);

}

return digits;

}

**// Test case**

const digits = [1, 2, 3];

console.log(plusOne(digits));

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

**6.**

function singleNumber(nums) {

let result = 0;

for (let num of nums) {

result ^= num;

}

return result;

}

**// Test case**

const nums = [2, 2, 1];

console.log(singleNumber(nums));

**Output:** 1

**7.**

function findMissingRanges(nums, lower, upper) {

const ranges = [];

let prev = lower - 1; // Tracking previous number

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

const curr = (i < nums.length) ? nums[i] : upper + 1;

if (curr - prev > 1) {

// There is a missing range between prev and curr

ranges.push(getRange(prev + 1, curr - 1));

}

prev = curr; // Update the previous number

}

return ranges;

}

function getRange(start, end) {

return (start === end) ? `${start}` : `${start}->${end}`;

}

// Test case

const nums = [0, 1, 3, 50, 75];

const lower = 0;

const upper = 99;

console.log(findMissingRanges(nums, lower, upper));

**Output:** [[2,2], [4,49], [51,74], [76,99]]

**8.**

function canAttendMeetings(intervals) {

intervals.sort((a, b) => a[0] - b[0]); // Sorting the intervals

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

const prevEnd = intervals[i - 1][1];

const currStart = intervals[i][0];

if (currStart < prevEnd) {

return false;

}

}

// No overlaps

return true;

}

**// Test case**

const intervals = [[0, 30], [5, 10], [15, 20]];

console.log(canAttendMeetings(intervals));

**Output:** false