**ASSIGNMENT1**

**1.**

function twoSum(nums, target) {

const map = new Map();

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

const complement = target - nums[i];

if (map.has(complement)) {

return [map.get(complement), i];

}

map.set(nums[i], i);

}

return [];

}

// Test case

const nums = [2, 7, 11, 15];

const target = 9;

const result = twoSum(nums, target);

console.log(result);

**Time Complexity:** O(n)

**Space Complexity**: O(n)

**2**. function removeElement(nums, val) {

let k = 0; // Number of elements not equal to val

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

if (nums[i] !== val) {

nums[k] = nums[i]; // Move non-val element to the front of the array

k++;

}

}

return k;

}

// Test case

const nums = [3, 2, 2, 3];

const val = 3;

const result = removeElement(nums, val);

console.log(`Result: ${result}`);

console.log(`Modified nums: [${nums}]`);

**Time Complexity:** O(n)

**Space Complexity**: O(1)

3. Algorithm:

1. In this code, we use two pointers, left and right, to define the search range within the array. We repeatedly calculate the middle index using (left + right) / 2 and compare the value at the middle index with the target value.
2. If the value at the middle index is equal to the target, we return the middle index as the target is found.
3. If the value at the middle index is less than the target, we update the left pointer to mid + 1, indicating that the target must be in the right half of the search range.
4. If the value at the middle index is greater than the target, we update the right pointer to mid - 1, indicating that the target must be in the left half of the search range.
5. We repeat this process until the left pointer is greater than the right pointer, indicating that the target is not found in the array. In this case, we return the left pointer as the index where the target would be inserted to maintain the sorted order.
6. In the provided example, the output will be 2 since the target value 5 is found at index 2 in the array nums.

4.

function plusOne(digits) {

const n = digits.length;

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

if (digits[i] === 9) {

digits[i] = 0;

} else {

digits[i]++;

return digits;

}

}

digits.unshift(1);

return digits;

}

// Test case

const digits = [1, 2, 3];

const result = plusOne(digits);

console.log(result);

5.

function merge(nums1, m, nums2, n) {

let p1 = m - 1; // Pointer for nums1

let p2 = n - 1; // Pointer for nums2

let p = m + n - 1; // Pointer for merged array

while (p1 >= 0 && p2 >= 0) {

if (nums1[p1] >= nums2[p2]) {

nums1[p] = nums1[p1];

p1--;

} else {

nums1[p] = nums2[p2];

p2--;

}

p--;

}

// Copy remaining elements from nums2 to nums1

while (p2 >= 0) {

nums1[p] = nums2[p2];

p--;

p2--;

}

}

// Test case

const nums1 = [1, 2, 3, 0, 0, 0];

const m = 3;

const nums2 = [2, 5, 6];

const n = 3;

merge(nums1, m, nums2, n);

console.log(nums1);

6.

function containsDuplicate(nums) {

const set = new Set();

for (let num of nums) {

if (set.has(num)) {

return true;

}

set.add(num);

}

return false;

}

// Test case

const nums = [1, 2, 3, 1];

const result = containsDuplicate(nums);

console.log(result);

7.

function moveZeroes(nums) {

let slow = 0;

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

if (nums[fast] !== 0) {

nums[slow] = nums[fast];

slow++;

}

}

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

nums[i] = 0;

}

}

// Test case

const nums = [0, 1, 0, 3, 12];

moveZeroes(nums);

console.log(nums);

8.

function findErrorNums(nums) {

const n = nums.length;

const count = new Map();

let duplicate, missing;

for (let num of nums) {

count.set(num, (count.get(num) || 0) + 1);

}

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

if (count.has(i)) {

if (count.get(i) === 2) {

duplicate = i;

}

} else {

missing = i;

}

}

return [duplicate, missing];

}

// Test case

const nums = [1, 2, 2, 4];

const result = findErrorNums(nums);

console.log(result);