Q1. function arrayPairSum(nums) {

//array sort

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

let sum = 0;

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

sum += nums[i];

}

return sum;

}

const nums = [1, 4, 3, 2];

const maxSum = arrayPairSum(nums);

console.log(maxSum);

Q2. function f(candyT) {

const n = candyT.length;

const half = n / 2;

const uniqueTypes = new Set(candyT);

let maxCount = 0;

function fun(currIndex, currCombination) {

if (currCombination.length === half) {

const uniqueCount = new Set(currCombination).size;

maxCount = Math.max(maxCount, uniqueCount);

return;

}

if (currIndex >= n) {

return;

}

//include

fun(currIndex + 1, [...currCombination, candyT[currIndex]]);

//exclude

fun(currIndex + 1, currCombination);

}

fun(0, []);

return maxCount;

}

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

const maxCand = f(candyT);

console.log(maxCand);

Q3. function findL(nums) {

const frequencyMap = new Map();

let longestSubsequenceLength = 0;

for (const num of nums) {

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

if (frequencyMap.has(num + 1)) {

const subsequenceLength = frequencyMap.get(num) + frequencyMap.get(num + 1);

longestSubsequenceLength = Math.max(longestSubsequenceLength, subsequenceLength);

}

if (frequencyMap.has(num - 1)) {

const subsequenceLength = frequencyMap.get(num) + frequencyMap.get(num - 1);

longestSubsequenceLength = Math.max(longestSubsequenceLength, subsequenceLength);

}

}

return longestSubsequenceLength;

}

const nums = [1,3,2,2,5,2,3,7];

const longestSubsequence = findL(nums);

console.log(longestSubsequence);

Q4. function f(dwd, n) {

let count = 0;

const length = fwd.length;

for (let i = 0; i < length && count < n; i++) {

if (fwd[i] === 0) {

const prev = i === 0 ? 0 : fwd[i - 1];

const next = i === length - 1 ? 0 : fwd[i + 1];

if (prev === 0 && next === 0) {

fwd[i] = 1;

count++;

}

}

}

return count >= n;

}

const fwd = [1, 0, 0, 0, 1];

const n = 1;

const canPlant = f(fwd, n);

console.log(canPlant);

Q5. function maxi(nums) {

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

const n = nums.length;

const possibility1 = nums[n - 1] \* nums[n - 2] \* nums[n - 3];

const possibility2 = nums[0] \* nums[1] \* nums[n - 1];

return Math.max(possibility1, possibility2);

}

const nums = [1, 2, 3];

const maxP = maxi(nums);

console.log(maxP);

Q6. function f(nums, rg) {

let left = 0;

let right = nums.length - 1;

while (left <= right) {

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

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

return mid;

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

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

const nums = [-1, 0, 3, 5, 9, 12];

const trg = 9;

const index = f(nums, trg);

console.log(index);

Q7. function mono(nums) {

let inc = true;

let dec = true;

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

if (nums[i] < nums[i - 1]) {

inc = false;

}

if (nums[i] > nums[i - 1]) {

dec = false;

}

}

return inc || dec;

}

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

console.log(mono(nums));

Q8. function f(nums, k) {

const min = Math.min(nums);

const max = Math.max(nums);

if (max - min <= 2 \* k) {

return 0;

}

const a1 = max - (min + k);

const a2 = (max - k) - min;

return Math.min(a1, a2);

}

const nums=[1], k = 0;

console.log(f(nums,k));