Assignment Questions 2

Question 1 Given an integer array nums of 2n integers, group these integers into n pairs (a1, b1), (a2, b2),..., (an, bn) such that the sum of min(ai, bi) for all i is maximized. Return the maximized sum.

Example 1: Input: nums = [1,4,3,2] Output: 4

Explanation: All possible pairings (ignoring the ordering of elements) are:

1. (1, 4), (2, 3) -> min(1, 4) + min(2, 3) = 1 + 2 = 3
2. (1, 3), (2, 4) -> min(1, 3) + min(2, 4) = 1 + 2 = 3
3. (1, 2), (3, 4) -> min(1, 2) + min(3, 4) = 1 + 3 = 4 So the maximum possible sum is 4

Answer→

```
function arrayPairSum(nums) {
  nums.sort((a, b) => a - b); // Sort the array in ascending order
  let pairSum = 0;
  for (let i = 0; i < nums.length; i += 2) {
    pairSum += nums[i];
  }
  return pairSum;
}

// Test case
const nums = [1, 4, 3, 2];
console.log(arrayPairSum(nums)); // Output: 4
```

Question 2 Alice has n candies, where the ith candy is of type candyType[i]. Alice noticed that she started to gain weight, so she visited a doctor.

The doctor advised Alice to only eat n / 2 of the candies she has (n is always even). Alice likes her candies very much, and she wants to eat the maximum number of different types of candies while still following the doctor's advice.

Given the integer array candyType of length n, return the maximum number of different types of candies she can eat if she only eats n / 2 of them.

Example 1: Input: candyType = [1,1,2,2,3,3] Output: 3 Explanation: Alice can only eat 6 / 2 = 3 candies. Since there are only 3 types, she can eat one of each type.

Answer→
```
function maxDifferentTypes(candyType) {
  const maxTypes = new Set(candyType).size; // Count the number of unique candy types
  const maxEaten = candyType.length / 2; // Maximum number of candies Alice can eat
  return Math.min(maxTypes, maxEaten);
}

// Test case
const candyType = [1, 1, 2, 2, 3, 3];
console.log(maxDifferentTypes(candyType)); // Output: 3
```

Question 3 We define a harmonious array as an array where the difference between its maximum value and its minimum value is exactly

1. Given an integer array nums, return the length of its longest harmonious subsequence among all its possible subsequences. A subsequence of an array is a sequence that can be derived from the array by deleting some or no elements without changing the order of the remaining elements.

Example 1: Input: nums = [1,3,2,2,5,2,3,7] Output: 5 Explanation: The longest harmonious subsequence is [3,2,2,2,3].

Answer→
```
function findLHS(nums) {
  const numFreq = new Map();
  let maxLength = 0;
  for (const num of nums) {
    numFreq.set(num, (numFreq.get(num) || 0) + 1);
  }
  for (const [num, freq] of numFreq) {
    if (numFreq.has(num + 1)) {
      maxLength = Math.max(maxLength, freq + numFreq.get(num + 1));
    }
  }
  return maxLength;
}

// Test case
const nums = [1, 3, 2, 2, 5, 2, 3, 7];
console.log(findLHS(nums)); // Output: 5
```

Question 4 You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in adjacent plots.
Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return true if n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule and false otherwise.

Example 1: Input: flowerbed = [1,0,0,0,1], n = 1 Output: true

Answer→

```
function canPlaceFlowers(flowerbed, n) {
  let count = 0;
  let i = 0;
  while (i < flowerbed.length) {
    if (flowerbed[i] === 0 && (i === 0 || flowerbed[i - 1] === 0) && (i === flowerbed.length - 1 ||
flowerbed[i + 1] === 0)) {
      flowerbed[i] = 1;
      count++;
    }
    i++;
  }
  return count >= n;
}

// Test case
const flowerbed = [1, 0, 0, 0, 1];
const n = 1;
console.log(canPlaceFlowers(flowerbed, n)); // Output: true
```

Question 5 Given an integer array nums, find three numbers whose product is maximum and return the maximum product.

Example 1: Input: nums = [1,2,3] Output: 6

Answer→

```
function maximumProduct(nums) {
  nums.sort((a, b) => a - b); // Sort the array in ascending order
  const n = nums.length;
  // Maximum product can be either the product of the three largest elements or the product of
the two smallest elements and the largest element
  return Math.max(nums[0] * nums[1] * nums[n - 1], nums[n - 3] * nums[n - 2] * nums[n - 1]);
}
```

```
// Test case
const nums = [1, 2, 3];
console.log(maximumProduct(nums)); // Output: 6
```

Question 6 Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums.

If target exists, then return its index. Otherwise, return -1. You must write an algorithm with O(log n) runtime complexity.

Input: nums = [-1,0,3,5,9,12], target = 9 Output: 4

Explanation: 9 exists in nums and its index is 4

Answer→

```
function binarySearch(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; // Target found
    } else if (nums[mid] < target) {
      left = mid + 1; // Search the right half
    } else {
      right = mid - 1; // Search the left half
    }
  }
  return -1; // Target not found
}

// Test case
const nums = [-1, 0, 3, 5, 9, 12];
const target = 9;
console.log(binarySearch(nums, target)); // Output: 4
```

Question 7 An array is monotonic if it is either monotone increasing or monotone decreasing. An array nums is monotone increasing if for all i <= j, nums[i] <= nums[j].

An array nums is monotone decreasing if for all i <= j, nums[i] >= nums[j]. Given an integer array nums, return true if the given array is monotonic, or false otherwise.

Example 1: Input: nums = [1,2,2,3] Output: true

Answer→
```
function isMonotonic(nums) {
  let increasing = true;
  let decreasing = true;
  for (let i = 1; i < nums.length; i++) {
    if (nums[i] < nums[i - 1]) {
      increasing = false;
    }
    if (nums[i] > nums[i - 1]) {
      decreasing = false;
    }
  }
  return increasing || decreasing;
}

// Test case
const nums = [1, 2, 2, 3];
console.log(isMonotonic(nums)); // Output: true
```

Question 8 You are given an integer array nums and an integer k. In one operation, you can choose any index i where 0 <= i < nums.length and change nums[i] to nums[i] + x where x is an integer from the range [-k, k]. You can apply this operation at most once for each index i.

The score of nums is the difference between the maximum and minimum elements in nums. Return the minimum score of nums after applying the mentioned operation at most once for each index in it.

Example 1: Input: nums = [1], k = 0 Output: 0 Explanation: The score is max(nums) - min(nums) = 1 - 1 = 0.

Answer→
```
function minScore(nums, k) {
  let minNum = Math.min(...nums);
  let maxNum = Math.max(...nums);
  if (maxNum - minNum <= k) {
    return maxNum - minNum;
  }
  const count = new Array(maxNum - minNum + 1).fill(0);
  for (const num of nums) {
    count[num - minNum]++;
  }
  let minScore = Infinity;
  let left = 0;
  let right = count.length - 1;
  let leftSum = 0;
  let rightSum = nums.length;
  while (left < right) {
    while (count[left] === 0) {
      left++;
      leftSum += left;
    }
    while (count[right] === 0) {
      right--;
      rightSum -= right;
    }
    if (leftSum < rightSum) {
      minScore = Math.min(minScore, (rightSum - leftSum) - (right - left));
      leftSum += left;
      count[left]--;
    } else {
      minScore = Math.min(minScore, (rightSum - leftSum) - (right - left));
      rightSum -= right;
      count[right]--;
    }
  }
  return minScore;
}

// Test case
const nums = [1];
const k = 0;
console.log(minScore(nums, k)); // Output: 0
```