

<aside>

💡 ****Question 1****

Given an integer array `nums` of $2n$ integers, group these integers into n pairs $(a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)$ such that the sum of $\min(a_i, b_i)$ 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. $(1, 4), (2, 3) \rightarrow \min(1, 4) + \min(2, 3) = 1 + 2 = 3$

2. $(1, 3), (2, 4) \rightarrow \min(1, 3) + \min(2, 4) = 1 + 2 = 3$

3. $(1, 2), (3, 4) \rightarrow \min(1, 2) + \min(3, 4) = 1 + 3 = 4$

So the maximum possible sum is 4

</aside>

```
class Solution {  
    public int arrayPairSum(int[] nums) {  
        Arrays.sort(nums);  
        int len = nums.length;  
        int result = 0;  
        for (int i = 0; i < len - 1; i += 2) {  
            result += nums[i];  
        }  
        return result;  
    }  
}
```

Question 2

Alice has n candies, where the i th 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.

```
class Solution {  
    public int distributeCandies(int[] candyType) {  
  
        Set<Integer> uniqueCandiesSet = new HashSet<>();  
        for (int candy: candyType) {  
            uniqueCandiesSet.add(candy);  
        }  
        return Math.min(uniqueCandiesSet.size(), candyType.length / 2);  
    }  
}
```

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 = [1,3,2,2,2,5,2,3,7]`

Output: 5

Explanation: The longest harmonious subsequence is `[3,2,2,2,3]`.

```
class Solution {
    public int findLHS(int[] nums) {
        int len = nums.length;
        int left = 0;
        int result = 0;
        Arrays.sort(nums);
        for (int i = 0; i < len; i++) {
            while (left < i && nums[i] - nums[left] > 1) {
                left++;
            }
            if (nums[i] - nums[left] == 1) {
                result = Math.max(i - left + 1, result);
            }
        }
        return result;
    }
}
```

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

```
class Solution {  
    public boolean canPlaceFlowers(int[] flowerbed, int n) {  
        for (int i = 0; i < flowerbed.length; i++) {  
            if (flowerbed[i] == 0 &&  
                (i == 0 || flowerbed[i - 1] == 0) &&  
                (i == flowerbed.length - 1 || flowerbed[i + 1] == 0)) {  
  
                flowerbed[i] = 1;  
                n--;  
            }  
  
            if (n <= 0) {  
                return true;  
            }  
        }  
        return false;  
    }  
}
```

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

```
public class Solution {  
    public int maximumProduct(int[] nums) {  
        Arrays.sort(nums);  
        int option1 = nums[0] * nums[1] * nums[nums.length - 1];  
        int option2 = nums[nums.length - 1] * nums[nums.length - 2] * nums[nums.length - 3];  
        return Math.max(option1, option2);  
    }  
}
```

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

```
class Solution {
    public int search(int[] nums, int target) {
        int n = nums.length;
        int l = 0, h = n-1;
        int p = 0;
        while(l<=h){
            p = l+(h-l)/2;
            if(nums[p]==target)
                return p;
            if(target<nums[p])
                h = p-1;
            else
                l = p+1;
        }
        return -1;
    }
}
```

Question 7

An array is monotonic if it is either monotone increasing or monotone decreasing.

An array `nums` is monotone increasing if for all $i \leq j$, `nums[i] ≤ nums[j]`. An array `nums` is monotone decreasing if for all $i \leq j$, `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`

```
class Solution {
    public boolean isMonotonic(int[] nums) {
        boolean increasing = true;
        boolean decreasing = true;

        for (int i = 1; i < nums.length; ++i) {
            increasing &= nums[i] >= nums[i - 1];
            decreasing &= nums[i] <= nums[i - 1];
        }

        return increasing || decreasing;
    }
}
```

