

 **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

```
class Solution {
public:
    int arrayPairSum(vector<int>& nums) {
        sort(nums.begin(), nums.end());

        int sum = 0;

        for (int i = 0; i < nums.size(); i += 2) {
            sum += nums[i];
        }

        return sum;
    }
};
```

## Question 2

Alice has  $n$  candies, where the  $i$ th candy is of type  $\text{candyType}[i]$ . 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  $\text{candyType}$  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:  $\text{candyType} = [1,1,2,2,3,3]$

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(vector<int>& candyType) {

        int n = candyType.size();
        int a = n/2;

        set<int> s;

        for(int i =0 ;i<n;i++)
        {
            s.insert(candyType[i]);
        }

        int b = s.size();

        return min(a,b);

    }
};
```

### 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]`.

```
class Solution {
public:
    int findLHS(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int n = nums.size();
        int l = 0;
        int r = 1;
        int maxLength = 0;

        while (r < n) {
            if (nums[r] - nums[l] == 1) {
                maxLength = max(maxLength, r - l + 1);
                r++;
            } else if (nums[r] - nums[l] > 1) {
                l++;
            } else {
                r++;
            }
        }

        return maxLength;
    }
};
```

#### 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:
    bool canPlaceFlowers(vector<int>&flowerbed, int n)
    {
        for(int i=0; i<flowerbed.size(); i++)
        {
            if(flowerbed[i]==0)
            {
                if((i==0 || flowerbed[i-1]==0)&&(i==flowerbed.size()-1 ||
flowerbed[i+1]==0))
                {
                    n--;
                    flowerbed[i]=1;
                }
            }
        }
        return (n<=0);
    }
};
```

### 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

```
class Solution {
public:
    int maximumProduct(vector<int>& nums) {

        int s=nums.size();
        sort(nums.begin(),nums.end());
        int a= (nums[s-1]*nums[s-2]*nums[s-3]);
        int b= (nums[0]*nums[1]*nums[s-1]);
        return max(a,b);
    }
};
```

### 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(vector<int>& nums, int target) {

        bool x= binary_search(nums.begin(),nums.end(),target);

        if(x==true){
            auto it= lower_bound(nums.begin(),nums.end(),target);
            return (it-nums.begin());
        }
        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:
    bool isMonotonic(vector<int>& nums) {

        int n= nums.size();
        int c=0;
        int s=0;
        for(int i=1;i<n;i++){
            if(nums[i-1]>=nums[i])
                c=c+1;
        }
        for(int i=1;i<n;i++){
            if(nums[i-1]<=nums[i])
                s=s+1;
        }

        if((c==n-1) || (s==n-1))
            return 1;
        else
            return 0;
    }
};
```

### Question 8

You are given an integer array `nums` and an integer `k`.

In one operation, you can choose any index `i` where  $0 \leq i < \text{nums.length}$  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  $\max(\text{nums}) - \min(\text{nums}) = 1 - 1 = 0$ .

```
class Solution {
public:
    int smallestRangeI(vector<int>& nums, int k) {

        int n= nums.size();
        sort(nums.begin(),nums.end());

        int x = nums[n-1]-nums[0];
        if(x<2*k)
            return 0;

        else
            return ((nums[n-1]-k)-(nums[0]+k));

    }
};
```