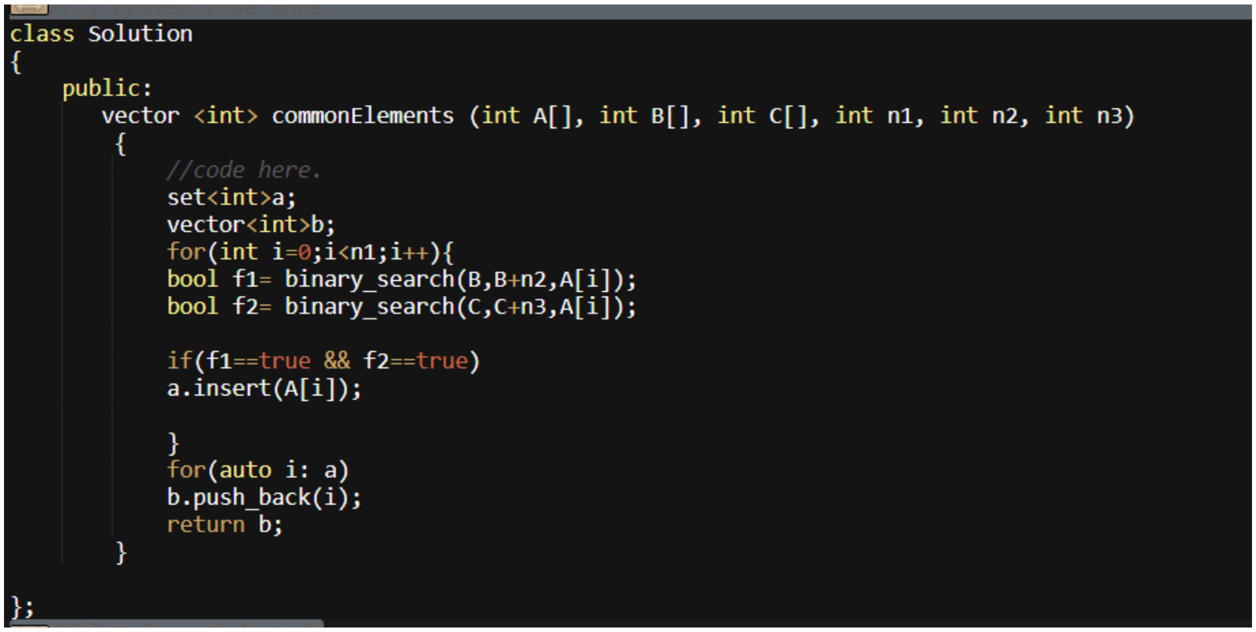
💡 **Question 1**
Given three integer arrays arr1, arr2 and arr3 **sorted** in **strictly increasing** order, return a sorted array of **only** the integers that appeared in **all** three arrays.

Example 1:
Input: arr1 = [1,2,3,4,5], arr2 = [1,2,5,7,9], arr3 = [1,3,4,5,8]

Output: [1,5]
Explanation: Only 1 and 5 appeared in the three arrays.



```
class Solution
{
    public:
       vector <int> commonElements (int A[], int B[], int C[], int n1, int n2, int n3)
        {
            //code here.
            set<int>a;
            vector<int>b;
            for(int i=0;i<n1;i++){
            bool f1= binary_search(B,B+n2,A[i]);
            bool f2= binary_search(C,C+n3,A[i]);

            if(f1==true && f2==true)
            a.insert(A[i]);

            }
            for(auto i: a)
            b.push_back(i);
            return b;
        }

};
```

💡 **Question 2**

Given two **0-indexed** integer arrays nums1 and nums2, return *a list* answer *of size* 2 *where:*

- answer[0] *is a list of all **distinct** integers in* nums1 *which are **not** present in* nums2*.*
- answer[1] *is a list of all **distinct** integers in* nums2 *which are **not** present in* nums1.

Note that the integers in the lists may be returned in **any** order.

Example 1:

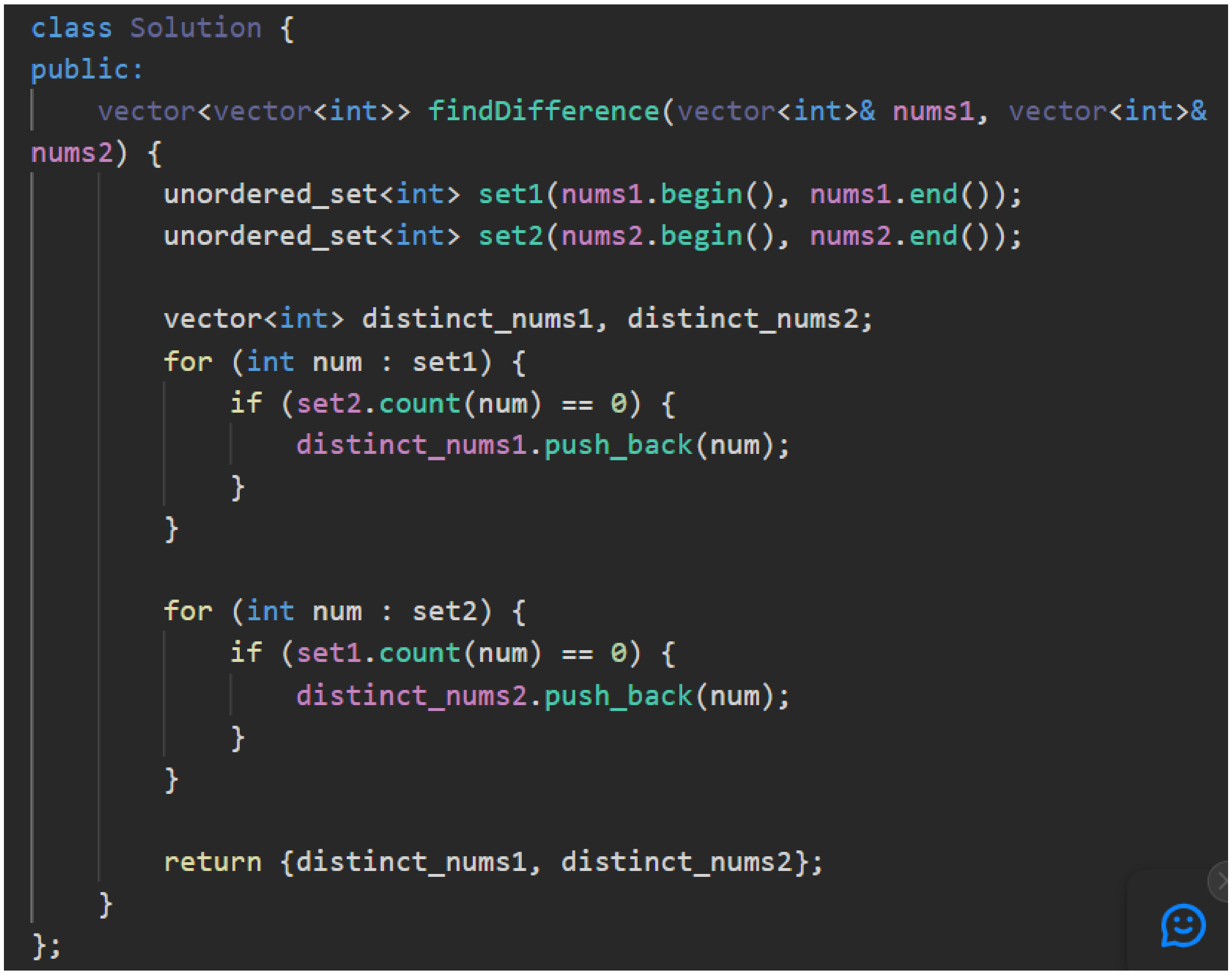
Input: nums1 = [1,2,3], nums2 = [2,4,6]

Output: [[1,3],[4,6]]

Explanation:

For nums1, nums1[1] = 2 is present at index 0 of nums2, whereas nums1[0] = 1 and nums1[2] = 3 are not present in nums2. Therefore, answer[0] = [1,3].

For nums2, nums2[0] = 2 is present at index 1 of nums1, whereas nums2[1] = 4 and nums2[2] = 6 are not present in nums2. Therefore, answer[1] = [4,6].



```
class Solution {
public:
    vector<vector<int>> findDifference(vector<int>& nums1, vector<int>&
nums2) {
        unordered_set<int> set1(nums1.begin(), nums1.end());
        unordered_set<int> set2(nums2.begin(), nums2.end());

        vector<int> distinct_nums1, distinct_nums2;
        for (int num : set1) {
            if (set2.count(num) == 0) {
                distinct_nums1.push_back(num);
            }
        }

        for (int num : set2) {
            if (set1.count(num) == 0) {
                distinct_nums2.push_back(num);
            }
        }

        return {distinct_nums1, distinct_nums2};
    }
};
```

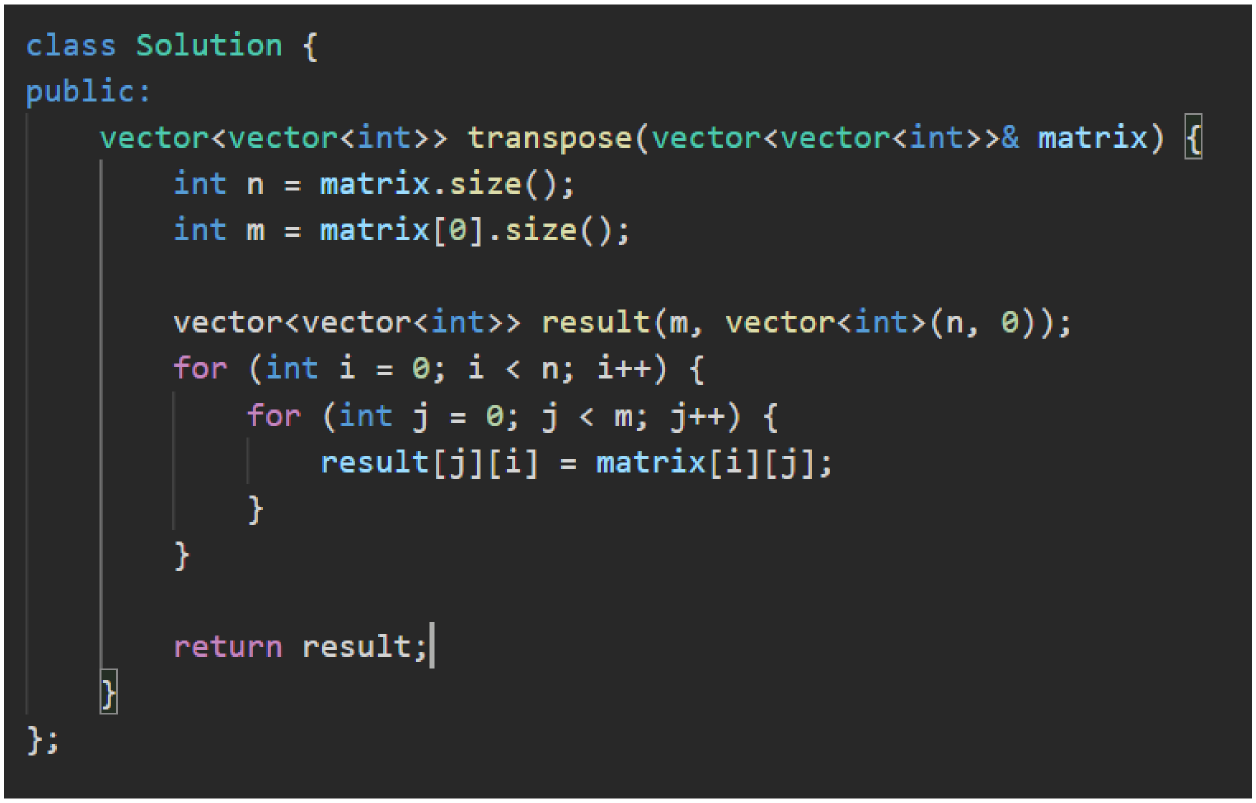
💡 **Question 3** Given a 2D integer array matrix, return *the **transpose** of* matrix.

The **transpose** of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.

Example 1:

Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [[1,4,7],[2,5,8],[3,6,9]]



```
class Solution {
public:
    vector<vector<int>> transpose(vector<vector<int>>& matrix) {
        int n = matrix.size();
        int m = matrix[0].size();

        vector<vector<int>> result(m, vector<int>(n, 0));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                result[j][i] = matrix[i][j];
            }
        }

        return result;
    }
};
```

💡 **Question 4** 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:

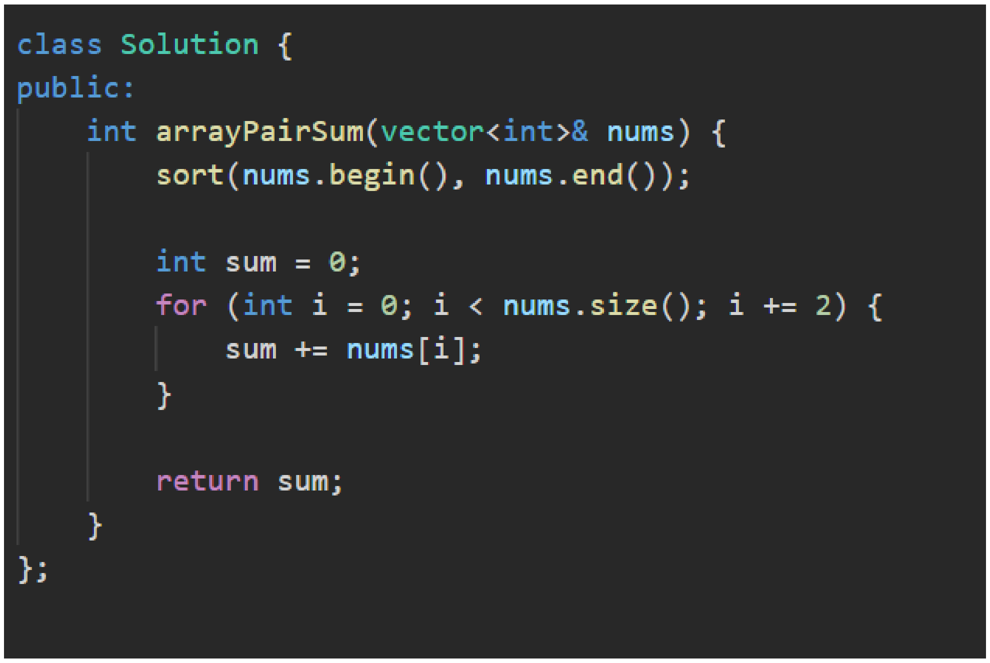
Input: nums = [1,4,3,2]

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.

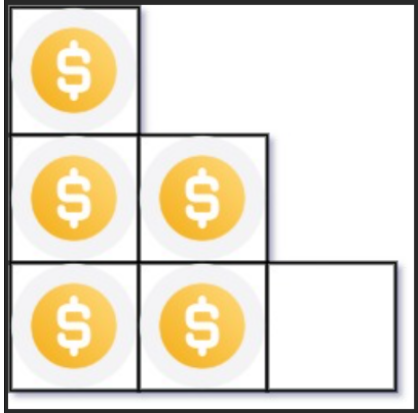


```cpp
class Solution {
public:
    int arrayPairSum(vector<int>& nums) {
        sort(nums.begin(), nums.end());

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

        return sum;
    }
};
```

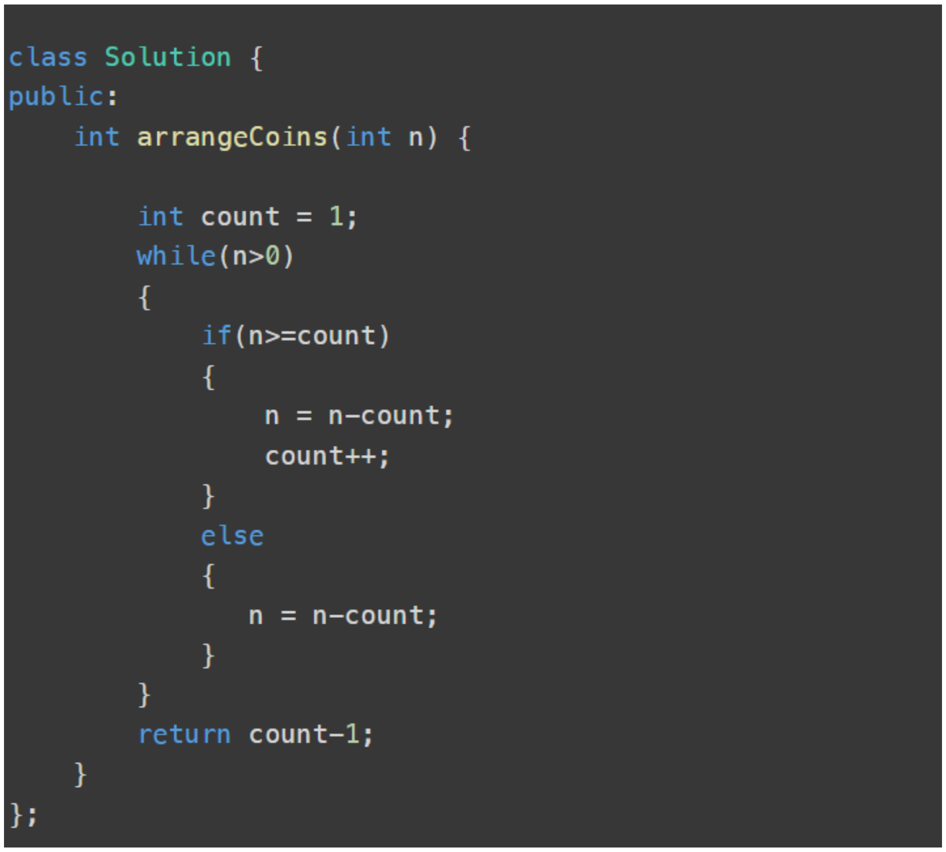
💡 **Question 5** You have n coins and you want to build a staircase with these coins. The staircase consists of k rows where the ith row has exactly i coins. The last row of the staircase **may be** incomplete.

Given the integer n, return *the number of **complete rows** of the staircase you will build*.

Example 1:



```
Input: n = 5
Output: 2
Explanation: Because the 3rd row is incomplete, we return 2.
```



```
class Solution {
public:
    int arrangeCoins(int n) {

        int count = 1;
        while(n>0)
        {
            if(n>=count)
            {
                n = n-count;
                count++;
            }
            else
            {
              n = n-count;
            }
        }
        return count-1;
    }
};
```

💡 **Question 6** Given an integer array nums sorted in **non-decreasing** order, return *an array of **the squares of each number** sorted in non-decreasing order*.

Example 1:

Input: nums = [-4,-1,0,3,10]

Output: [0,1,9,16,100]

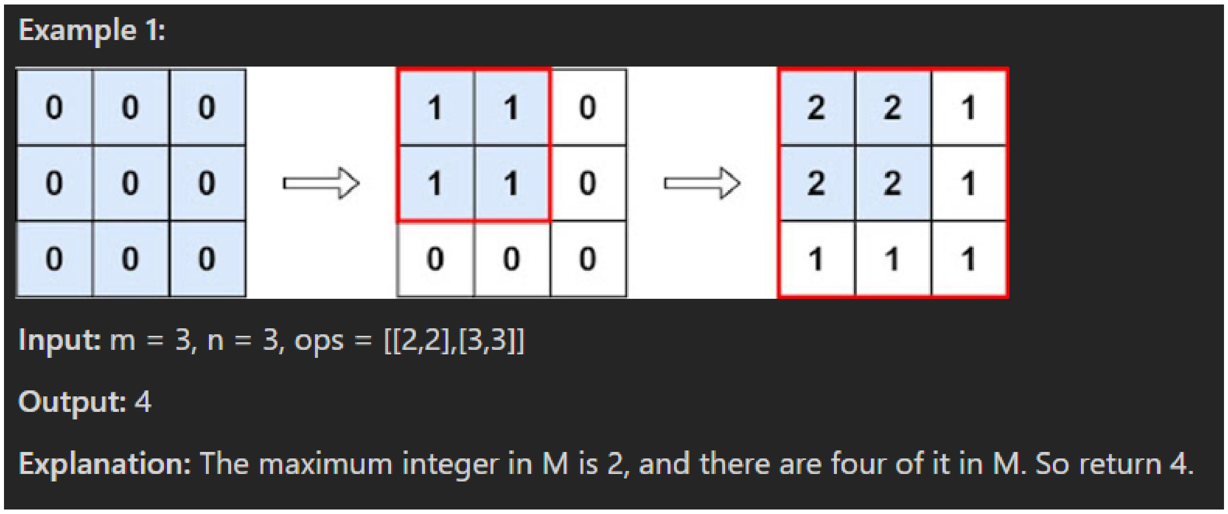
Explanation: After squaring, the array becomes [16,1,0,9,100]. After sorting, it becomes [0,1,9,16,100]

```cpp
class Solution {
public:
    vector<int> sortedSquares(vector<int>& nums) {
        int n=nums.size();

        for(int i=0;i<n;i++){
            nums[i]=nums[i]*nums[i];
        }
        sort(nums.begin(),nums.end());
        return nums;

    }
};
```

💡 **Question 7** You are given an m x n matrix M initialized with all 0's and an array of operations ops, where ops[i] = [ai, bi] means M[x][y] should be incremented by one for all 0 <= x < ai and 0 <= y < bi.

Count and return *the number of maximum integers in the matrix after performing all the operations*

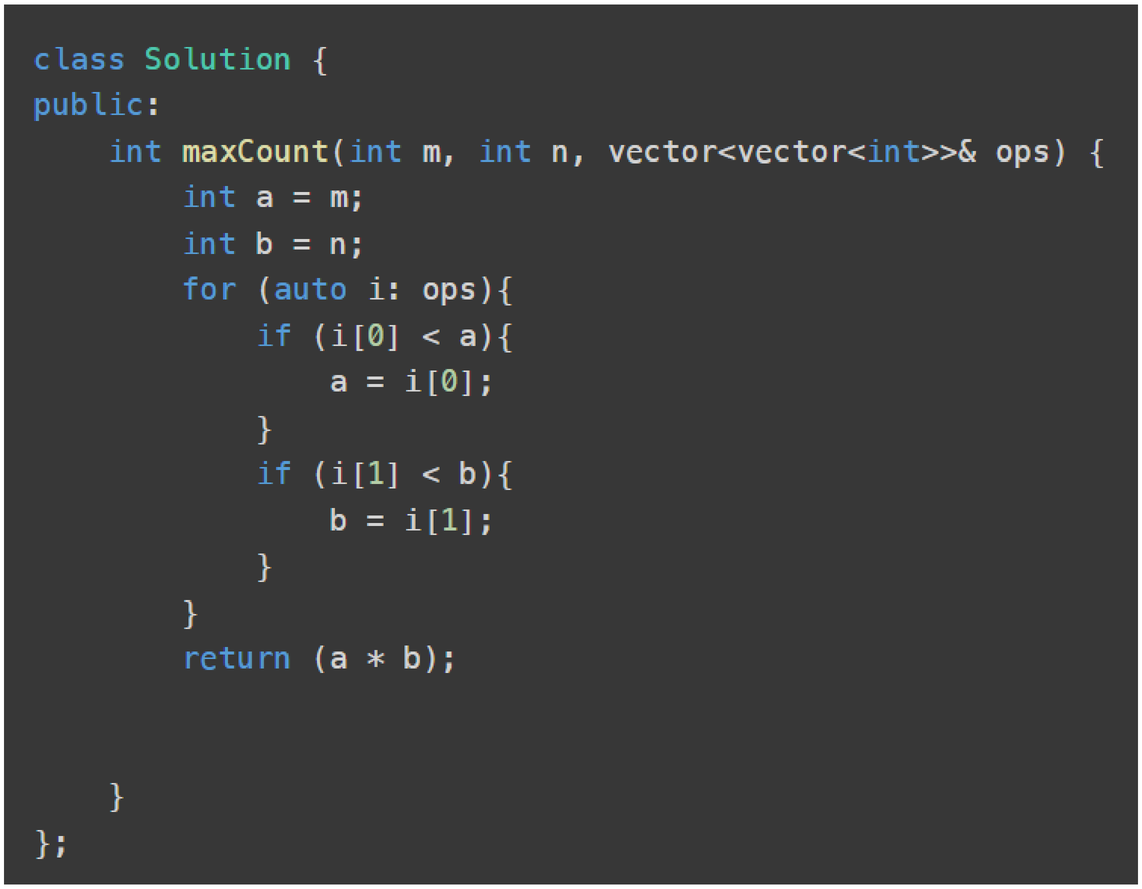
Example 1:


Example 1:

Input: m = 3, n = 3, ops = [[2,2],[3,3]]

Output: 4

Explanation: The maximum integer in M is 2, and there are four of it in M. So return 4.



```
class Solution {
public:
    int maxCount(int m, int n, vector<vector<int>>& ops) {
        int a = m;
        int b = n;
        for (auto i: ops){
            if (i[0] < a){
                a = i[0];
            }
            if (i[1] < b){
                b = i[1];
            }
        }
        return (a * b);


    }
};
```

💡 **Question 8**

Given the array nums consisting of 2n elements in the form [x1,x2,...,xn,y1,y2,...,yn].

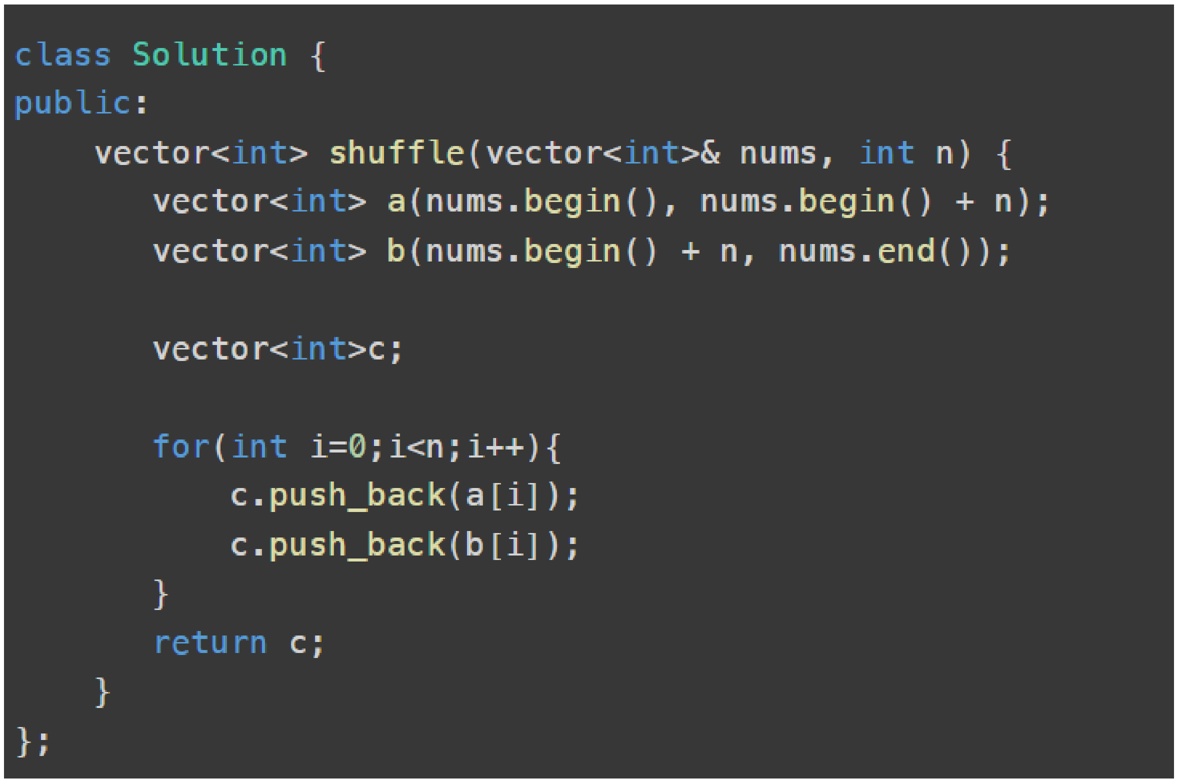
Return the array in the form [x1,y1,x2,y2,...,xn,yn].

Example 1:

Input: nums = [2,5,1,3,4,7], n = 3

Output: [2,3,5,4,1,7]

Explanation: Since x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 then the answer is [2,3,5,4,1,7].



```
class Solution {
public:
    vector<int> shuffle(vector<int>& nums, int n) {
       vector<int> a(nums.begin(), nums.begin() + n);
       vector<int> b(nums.begin() + n, nums.end());

       vector<int>c;

       for(int i=0;i<n;i++){
           c.push_back(a[i]);
           c.push_back(b[i]);
       }
       return c;
    }
};
```