```java
/*Problem 1:
There are n kids with candies. You are given an integer array candies, where
each candies[i] represents the number of
candies the i
th
kid has, and an integer extraCandies, denoting the number of extra candies
that you have.
Return a boolean array result of length n, where result[i] is true if, after
giving the i
th
kid all the extraCandies,
they will have the greatest number of candies among all the kids, or false
otherwise.
Note that multiple kids can have the greatest number of candies.*/

class Solution {
    public List<Boolean> kidsWithCandies(int[] candies, int extraCandies) {
        // Initialize maximum element
         int max = candies[0];
         // Traverse array elem to find the highest number
         for (int i = 1; i < candies.length; i++)
             if (candies[i] > max)
                 max = candies[i];
        //Initialize output list
        List<Boolean> output = new ArrayList<>();
        //Loop through each elem to set output[i] to true or false, depending
on the sum of candies[i] and extraCandies
        for(int i =0; i<candies.length; i++){
            if(candies[i]+extraCandies<max){
                output.add(false);
            }else{
                output.add(true);
            }
        }
        //finally, return the output list
        return output;
    }
}
```

```cpp
/*Problem 2:
You are given an integer array height of length n. There are n vertical lines
drawn such that the two endpoints of
the i
th
line are (i, 0) and (i, height[i]).
Find two lines that together with the x-axis form a container, such that the
container contains the most water.
Return the maximum amount of water a container can store.
Notice that you may not slant the container.*/

Class Solution {
public:
    int maxArea(vector<int>& H) {
        int ans = 0, i = 0, j = H.size()-1, res = 0;
        while (i < j) {
            if (H[i] <= H[j]) {
                res = H[i] * (j - i);
                i++;
            } else {
                res = H[j] * (j - i);
                j--;
            }
            if (res > ans) ans = res;
        }
        return ans;
    }
};
```

```java
/*Problem 3:
Given an integer array nums of length n and an integer target, find three
integers in nums such that the sum is closest to
target.
Return the sum of the three integers.
You may assume that each input would have exactly one solution. */
public class ThreeSumClosest {

    public int threeSumClosest(int[] nums, int target) {
        // Sort the array
        Arrays.sort(nums);
        // Length of the array
        int n = nums.length;
        // Result
        int closest = nums[0] + nums[1] + nums[n - 1];
        // Loop for each element of the array
        for (int i = 0; i < n - 2; i++) {
            // Left and right pointers
            int j = i + 1;
            int k = n - 1;
            // Loop for all other pairs
            while (j < k) {
                int sum = nums[i] + nums[j] + nums[k];
                if (sum <= target) {
                    j++;
                } else {
                    k--;
                }
                if (Math.abs(closest - target) > Math.abs(sum - target)) {
                    closest = sum;
                }
            }
        }
        return closest;
    }
}
```

```
/* Problem 4:
A permutation of an array of integers is an arrangement of its members into a
sequence or linear order.
● For example, for arr = [1,2,3], the following are all the permutations of
arr: [1,2,3], [1,3,2], [2, 1,
3], [2, 3, 1], [3,1,2], [3,2,1].
The next permutation of an array of integers is the next lexicographically
greater permutation of its integer. More formally, if
all the permutations of the array are sorted in one container according to
their lexicographical order, then the next
permutation of that array is the permutation that follows it in the sorted
container. If such arrangement is not possible, the
array must be rearranged as the lowest possible order (i.e., sorted in
ascending order).
●
●
●
For example, the next permutation of arr = [1,2,3] is [1,3,2].
Similarly, the next permutation of arr = [2,3,1] is [3,1,2].
While the next permutation of arr = [3,2,1] is [1,2,3] because [3,2,1] does
not have a lexicographical larger
rearrangement.
Given an array of integers nums, find the next permutation of nums.
The replacement must be in place and use only constant extra memory.*/

class Solution {
    public void nextPermutation(int[] nums) {
        int index = -1,n = nums.length;
        for(int i=n-2;i>=0;i--){
            if(nums[i]<nums[i+1]){
                index = i;
                break;
            }
        }
        for(int i=n-1;i>=0 && index!=-1;i--){
            if(nums[i]>nums[index]){
                int temp = nums[index];
                nums[index] = nums[i];
                nums[i] = temp;
                break;
            }
        }
        int l = index + 1,r = n - 1;
        while(l<r){
            int temp = nums[l];
            nums[l] = nums[r];
            nums[r] = temp;
```

```
            l++;r--;
        }
    }
}



/*Problem 5:
There is an integer array nums sorted in ascending order (with distinct
values).
Prior to being passed to your function, nums is possibly rotated at an unknown
pivot index k (1 <= k < nums.length)
such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0],
nums[1], ..., nums[k1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be
rotated at pivot index 3 and become [4,5,6,7,0,1,2].
Given the array nums after the possible rotation and an integer target, return
the index of target if it is in nums, or -1 if it
is not in nums.
You must write an algorithm with O(log n) runtime complexity.*/
import java.util.*;
import java.io.*;

public class Main {

    public static int find(int[]arr,int target) {
        //write your code here
    }

    public static void main(String[]args) {
        //input work
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int[]arr = new int[n];

        for(int i=0; i < n;i++) {
            arr[i] = scn.nextInt();
        }

        int target = scn.nextInt();
        int ans = find(arr,target);
        System.out.println(ans);
    }
}
```

```java
/*Problem 6:
Given a collection of numbers, nums, that might contain duplicates,
 return all possible unique permutations in any order.*/
 class Solution {
    public List<List<Integer>> permuteUnique(int[] nums) {
        List<List<Integer>> res = new LinkedList<>();
        if (nums == null || nums.length == 0) {
            return res;
        }
        Arrays.sort(nums);
        boolean[] visited = new boolean[nums.length];
        dfs(nums, visited, new LinkedList<Integer>(), res);
        return res;
    }
    private void dfs(int[] nums, boolean[] visited, List<Integer> curr,
List<List<Integer>> res) {
        if (curr.size() == nums.length) {
            res.add(new LinkedList<Integer>(curr));
            return;
        }
        for (int i = 0; i < nums.length; i ++) {
            if (visited[i] == true) {
                continue;
            }
            if (i == 0 || nums[i] != nums[i - 1] || (nums[i] == nums[i - 1] &&
visited[i - 1] == true)) {
                visited[i] = true;
                curr.add(nums[i]);
                dfs(nums, visited, curr, res);
                curr.remove(curr.size() - 1);
                visited[i] = false;
            }
        }
    }
}
```

```java
/*Problem 7:
Given an integer array nums, find the subarray with the largest sum, and
return its sum. */
class Solution {
    public List<List<Integer>> permuteUnique(int[] nums) {
        List<List<Integer>> res = new LinkedList<>();
        if (nums == null || nums.length == 0) {
            return res;
        }
        Arrays.sort(nums);
        boolean[] visited = new boolean[nums.length];
        dfs(nums, visited, new LinkedList<Integer>(), res);
        return res;
    }
    private void dfs(int[] nums, boolean[] visited, List<Integer> curr,
List<List<Integer>> res) {
        if (curr.size() == nums.length) {
            res.add(new LinkedList<Integer>(curr));
            return;
        }
        for (int i = 0; i < nums.length; i ++) {
            if (visited[i] == true) {
                continue;
            }
            if (i == 0 || nums[i] != nums[i - 1] || (nums[i] == nums[i - 1] &&
visited[i - 1] == true)) {
                visited[i] = true;
                curr.add(nums[i]);
                dfs(nums, visited, curr, res);
                curr.remove(curr.size() - 1);
                visited[i] = false;
            }
        }
    }
}
```

```cpp
/*Problem 8
Given an array of intervals where intervals[i] = [starti, endi], merge all
overlapping intervals, and return an
array of the non-overlapping intervals that cover all the intervals in the
input.*/
bool isOverlap(int minS, int maxE, vector<int> interval)
{
    if (minS > interval[1] || maxE < interval[0])
    {
        return false;
    }

    return true;
}

vector<vector<int>> mergeIntervals(vector<vector<int>> &intervals)
{
    int n = intervals.size();
    vector<vector<int>> res;

    vector<bool> vis(n, false);

    for (int i = 0; i < n; i++)
    {
        if (vis[i])
        {
            continue;
        }

        vis[i] = true;
        int minS = intervals[i][0];
        int maxE = intervals[i][1];

        while (true)
        {
            int cnt = 0;

            for (int j = 0; j < n; j++)
            {
                if (!vis[j] && isOverlap(minS, maxE, intervals[j]))
                {
                    vis[j] = true;
                    minS = min(minS, intervals[j][0]);
                    maxE = max(maxE, intervals[j][1]);
                    cnt++;
                }
            }
```

```
            if (cnt == 0)
            {
                break;
            }
        }

        vector<int> interval = {minS, maxE};
        res.push_back(interval);
    }

    sort(res.begin(), res.end());
    return res;
}
```

```
/* Problem 9:
Given an array nums with n objects colored red, white, or blue, sort them in-
place so that objects of the same color are
adjacent, with the colors in the order red, white, and blue.
We will use the integers 0, 1, and 2 to represent the color red, white, and
blue, respectively.
You must solve this problem without using the library's sort function.*/

#include <bits/stdc++.h>
using namespace std;

// Function to sort the input array,
// the array is assumed
// to have values in {0, 1, 2}
void sort012(int a[], int arr_size)
{
    int lo = 0;
    int hi = arr_size - 1;
    int mid = 0;

    // Iterate till all the elements
    // are sorted
    while (mid <= hi) {
        switch (a[mid]) {

        // If the element is 0
        case 0:
            swap(a[lo++], a[mid++]);
            break;
```

```
        // If the element is 1 .
        case 1:
            mid++;
            break;

        // If the element is 2
        case 2:
            swap(a[mid], a[hi--]);
            break;
        }
    }
}

// Function to print array arr[]
void printArray(int arr[], int arr_size)
{
    // Iterate and print every element
    for (int i = 0; i < arr_size; i++)
        cout << arr[i] << " ";
}

// Driver Code
int main()
{
    int arr[] = { 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);

    sort012(arr, n);

    printArray(arr, n);

    return 0;
}
```

```
/* Problem 10
You are given an integer array prices where prices[i] is the price of a given
stock on the i
th day.
On each day, you may decide to buy and/or sell the stock. You can only hold at
most one share of the stock at any time.
However, you can buy it then immediately sell it on the same day.
Find and return the maximum profit you can achieve. */
class Solution {
public:
    int maxProfit(vector<int>& prices) {
```

```
        int sum = 0;
        for(int i=0;i<prices.size()-1;i++)
        {
            if(prices[i+1]>prices[i])
                sum = sum + prices[i+1]-prices[i];
        }
        return sum;
    }
};
```

```
/* Problem 11
Given a 0-indexed integer array nums, find a peak element, and return its
index. If the array contains multiple peaks, return
the index to any of the peaks.*/
class Solution {
    public int findPeakElement(int[] nums) {
        int left = 0, right = nums.length - 1;
        while (left < right) {
            int mid = (left + right) >> 1;
            if (nums[mid] > nums[mid + 1]) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }
        return left;
    }
```

```
/* Problem 12
Given two sorted arrays nums1 and nums2 of size m and n respectively,
 return the median of the two sorted arrays.*/
 #include<bits/stdc++.h>
using namespace std;

float median(int nums1[],int nums2[],int m,int n) {
    int finalArray[n+m];
    int i=0,j=0,k=0;
    while(i<m && j<n) {
        if(nums1[i]<nums2[j]) {
            finalArray[k++] = nums1[i++];
        }
        else {
            finalArray[k++] = nums2[j++];
        }
```

```
    }
    if(i<m) {
        while(i<m)
            finalArray[k++] = nums1[i++];
    }
    if(j<n) {
        while(j<n)
            finalArray[k++] = nums2[j++];
    }
    n = n+m;
    if(n%2==1)
        return finalArray[((n+1)/2)-1];
    else return ((float)finalArray[(n/2)-1]+(float)finalArray[(n/2)])/2;
}

int main() {
    int nums1[] = {1,4,7,10,12};
    int nums2[] = {2,3,6,15};
    int m = sizeof(nums1)/sizeof(nums1[0]);
    int n = sizeof(nums2)/sizeof(nums2[0]);
    cout<<"The median of two sorted array is "<<fixed<<setprecision(5)
    <<median(nums1,nums2,m,n);
    return 0;
}
```

```
/* Problem 13
You are given an integer array height of length n. There are n vertical lines
drawn such that the two endpoints of the i
th
line are
(i, 0) and (i, height[i]).
Find two lines that together with the x-axis form a container, such that the
container contains the most water.
Return the maximum amount of water a container can store. */
#include <iostream>
using namespace std;

int maxContainer(int arr[], int n)
{
    int maxArea = 0;

        // Find the area for every pair of boundaries i and j
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
                    // Calculate the area
                    // and update the variable maxArea, if greater
```

```
                    maxArea = max(min(arr[j], arr[i]) * (j - i), maxArea);
        }
    }
    return maxArea;
}

// Driver code
int main()
{
    int height[] = {1, 8, 6, 2, 5, 4, 8, 3, 7};
    int height2[] = { 1, 1 };

    // Call the function for the first array
    int len1 = sizeof(height) / sizeof(height[0]);
    cout << maxContainer(height, len1);

    // Call the function for the second array
    int len2 = sizeof(height2) / sizeof(height2[0]);
    cout << endl << maxContainer(height2, len2);
}
```

```
/*Problem 14
Given a binary array nums, return the maximum number of consecutive 1's in the
array */
class Programming9{
public:
    int maxConsecutiveOnes(vector<int>& nums) {
        int count=0,result=0;
        int n=nums.size();
        for(int i=0;i<n;i++){
            if(nums[i]==1){
                count++;
                result=max(count,result);
            }
            else
                count=0;
        }
        return result;
    }
};
```

```
/* Problem 15
Given an integer array nums and an integer k, return the kth largest element
in the array.
Note that it is the kth largest element in the sorted order, not the kth
distinct element.
You must solve it in O(n) time complexity.*/
```

```cpp
#include<bits/stdc++.h>
using namespace std;

int KthLargestElement(vector<int> arr,int n,int k){
    sort(arr.begin(),arr.end());  // sorting the array
        return arr[n-k];          // return kth largest element
}

int main(){
    vector<int> arr{2,1,4,6,3,9,7};
    int n = arr.size();
    int k = 2;
    int x = KthLargestElement(arr,n,k);
    cout<<"Kth largest element is "<<x;
    return 0;
}
```