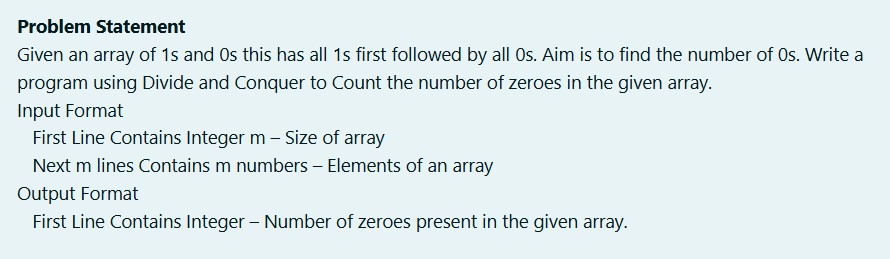
DIVIDE AND CONQUER

**QUESTION 4.A AIM:**



**ALGORITHM :**

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize array a of size n

Step 4: For each index i from 0 to n-1, input a[i]

Step 5: Call the function countz(a, 0, n - 1) and store its result in count

Step 6: Print the value of count

Step 7: Stop

Function **countz(a[], l, r)**:

Step 1: If l > r, return 0

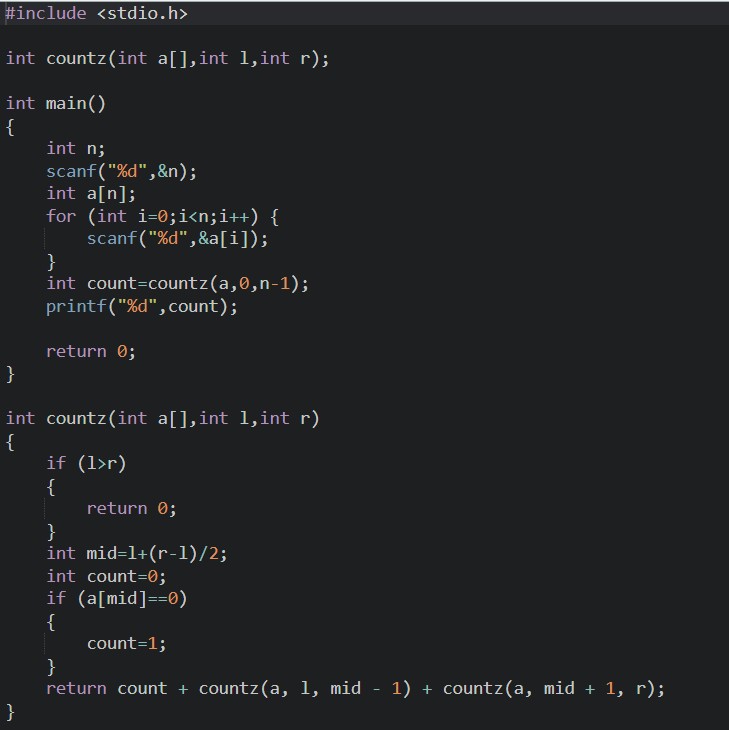
Step 2: Calculate mid as l + (r - l) / 2

Step 3: Initialize count to 0

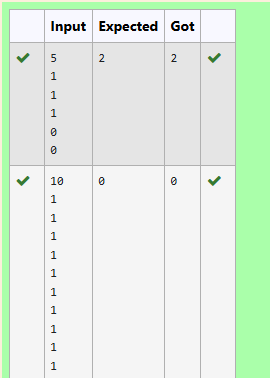
Step 4: If a[mid] == 0, set count = 1

Step 5: Return count + countz(a, l, mid - 1) + countz(a, mid + 1, r)

**PROGRAM :**



**OUTPUT:**

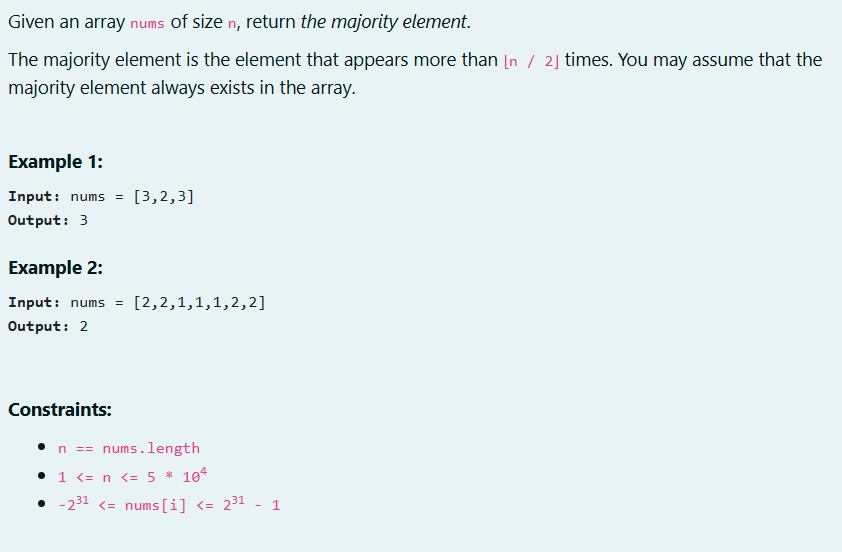


**RESULT :**

The above program is executed successfully .

**QUESTION 4.B**

**AIM :**



**ALOGORITHM :**

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize array a of size n

Step 4: For each index i from 0 to n-1, input a[i]

Step 5: Call the function majority(a, 0, n - 1) and store its result in majoele

Step 6: If majoele is not -1, print majoele; otherwise, print "No Majority Element"

Step 7: Stop

**Function majority(a[], l, r):**

Step 1: If l == r, return a[l]

Step 2: Calculate mid as (l + r) / 2

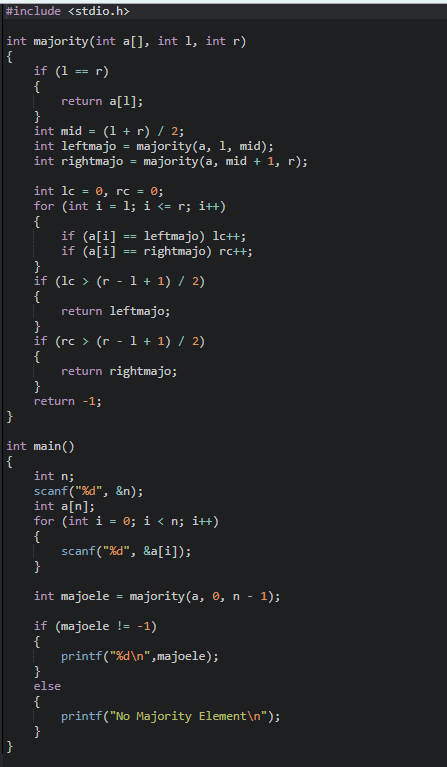
Step 3: Call majority(a, l, mid) and store its result in leftmajo Step 4: Call majority(a, mid + 1, r) and store its result in rightmajo Step 5: Initialize lc and rc to 0

Step 6: For each index i from l to r, if a[i] == leftmajo, increment lc; if a[i] == rightmajo, increment

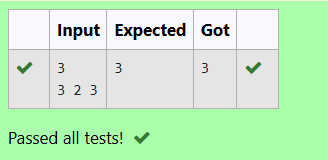
rc

Step 7: If lc > (r - l + 1) / 2, return leftmajo Step 8: If rc > (r - l + 1) / 2, return rightmajo Step 9: Return -1

**PROGRAM :**



**OUTPUT :**

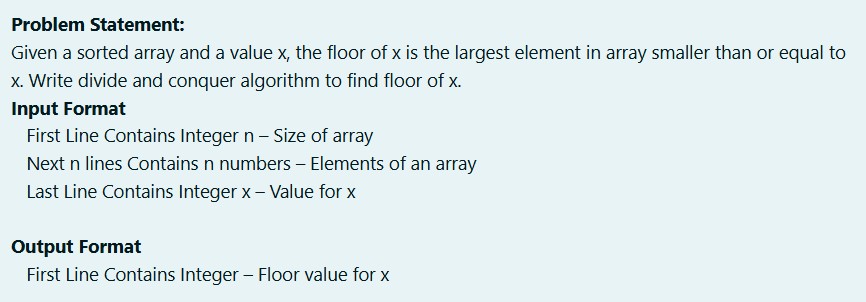


**RESULT :**

The above program is executed successfully.

**QUESTION 4.C**

**AIM :**



**ALGORITHM :**

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize array a of size n

Step 4: For each index i from 0 to n-1, input a[i]

Step 5: Input integer k

Step 6: Call findfloor(a, 0, n - 1, k)

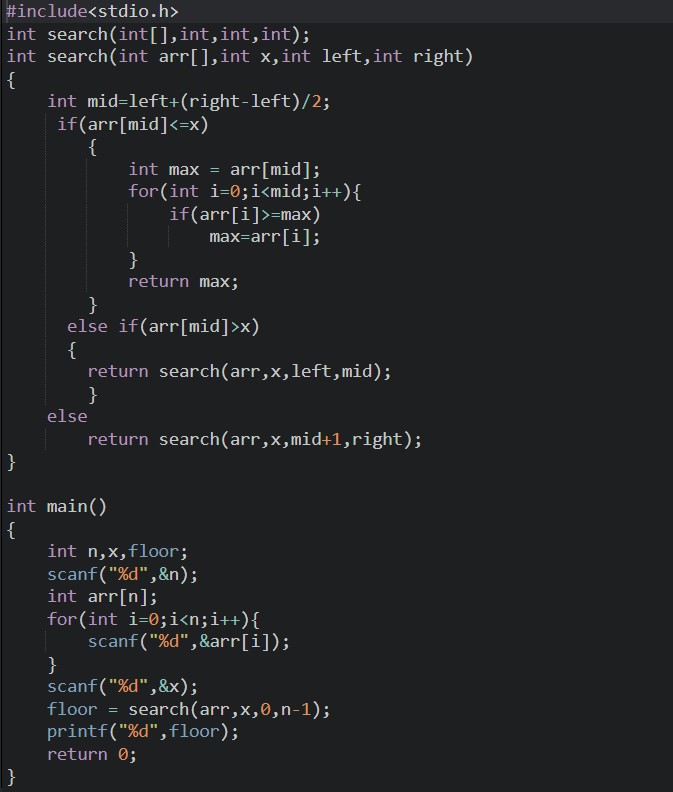
Step 7: Stop

**Function findfloor(a[], l, r, key):**

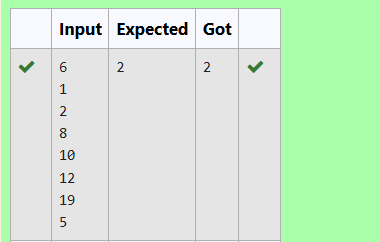
Step 1: If a[r] <= key, print a[r] and return Step 2: If l < r, do Steps 3 and 4

Step 3: Calculate mid as (l + r) / 2 Step 4: Call findfloor(a, mid + 1, r, key) Step 5: Call findfloor(a, l, mid, key)

**PROGRAM :**



**OUTPUT:**

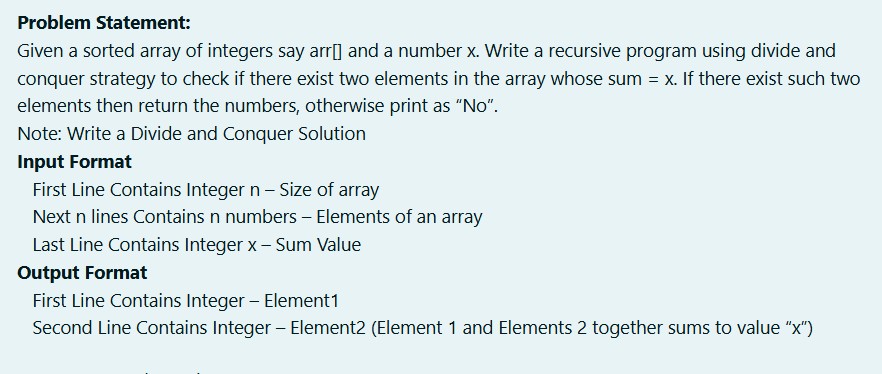


**RESULT:**

The above program is executed successfully.

**QUESTION 4.B**

**AIM :**



**ALGORITHM :**

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize array arr of size n

Step 4: For each index i from 0 to n-1, input arr[i]

Step 5: Input integer x

Step 6: Call findPair(arr, 0, n - 1, x)

Step 7: Stop

**Function findPair(arr[], left, right, x):**

Step 1: If left >= right, print "No" and return Step 2: Calculate sum as arr[left] + arr[right]

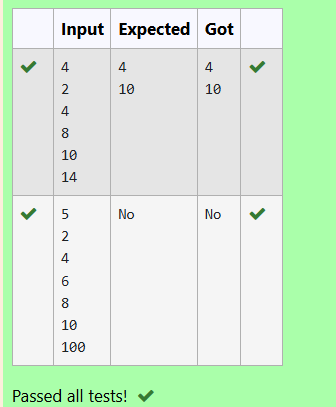
Step 3: If sum == x, print arr[left] and arr[right], and return Step 4: If sum < x, call findPair(arr, left + 1, right, x)

Step 5: Otherwise, call findPair(arr, left, right - 1, x)

**PROGRAM :**



**OUTPUT:**

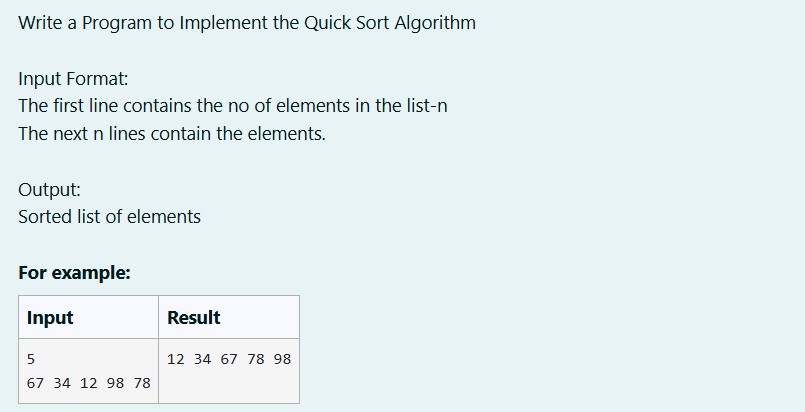


**RESULT:**

The above program is executed successfully.

**QUESTION 4.E**

**AIM:**



**ALGORITHM :**

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize array arr of size n

Step 4: For each index i from 0 to n-1, input arr[i]

Step 5: Call quickSort(arr, 0, n - 1)

Step 6: For each index i from 0 to n-1, print arr[i]

Step 7: Stop

**Function quickSort(arr[], left, right):** Step 1: If left < right, do Steps 2 to 7 Step 2: Set pivot to (left + right) / 2 Step 3: Initialize i to left and j to right

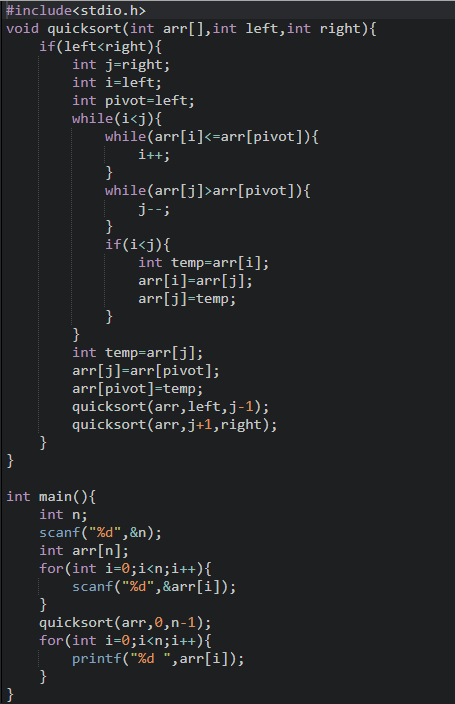
Step 4: While i < j, do Steps 5.1 to 5.4

Step 5.1: While arr[pivot] >= arr[i], increment i Step 5.2: While arr[pivot] < arr[j], decrement j Step 5.3: If i <= j, swap arr[i] and arr[j]

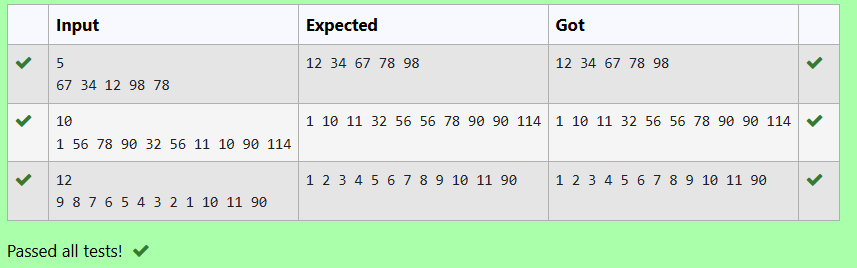
Step 6: Swap arr[j] and arr[pivot]

Step 7: Call quickSort(arr, left + 1, right)

**PROGRAM :**



**OUTPUT :**



**RESULT:**

The above program is executed successfully .