Question 1 | Correct Mark 1.00 out of 1.00

Problem Statement

Given an array of 1s and 0s this has all 1s first followed by all 0s. Aim is to find the number of 0s. Write a program using Divide and Conquer to Count the number of zeroes in the given array.

Input Format

First Line Contains Integer m – Size of array

Next m lines Contains m numbers – Elements of an array

Output Format

First Line Contains Integer – Number of zeroes present in the given array.

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int n;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    /*int l=0,r=n-1,ind=0;
    while(l<r)
    {
        int mid=(l+r)/2;
        if(arr[mid]==0)
        {
            ind=mid;
            r=mid-1;
        }
        else
        {
            l=mid+1;
        }
    }
    printf("%d",n-ind);
    return 0;*/
    int l=0,r=n-1,ind=0;
    while(l<r)
    {
        int mid=(l+r)/2;
        if(arr[mid]==0)
        {
            ind=mid;
            r=mid;
        }
        else if(arr[mid]!=0)
        {
            l=mid+1;
        }
    }
    if(arr[0]==1 && arr[n-1]==1)
    {
        printf("%d",0);
    }
    else if(ind!=-1)
    {
        printf("%d",n-ind);
    }
}
```

	Input	Expected	Got	
✓	5 1 1 1 0 0	2	2	✓
✓	10 1 1 1 1 1 1 1 1 1 1	0	0	✓
✓	8 0 0 0 0 0 0 0 0	8	8	✓
✓	17 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0	2	2	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 1 | Correct Mark 1.00 out of 1.00

Given an array `nums` of size `n`, return *the majority element*.

The majority element is the element that appears more than `⌊n / 2⌋` times. You may assume that the majority element always exists in the array.

Example 1:

```
Input: nums = [3,2,3]
Output: 3
```

Example 2:

```
Input: nums = [2,2,1,1,1,2,2]
Output: 2
```

Constraints:

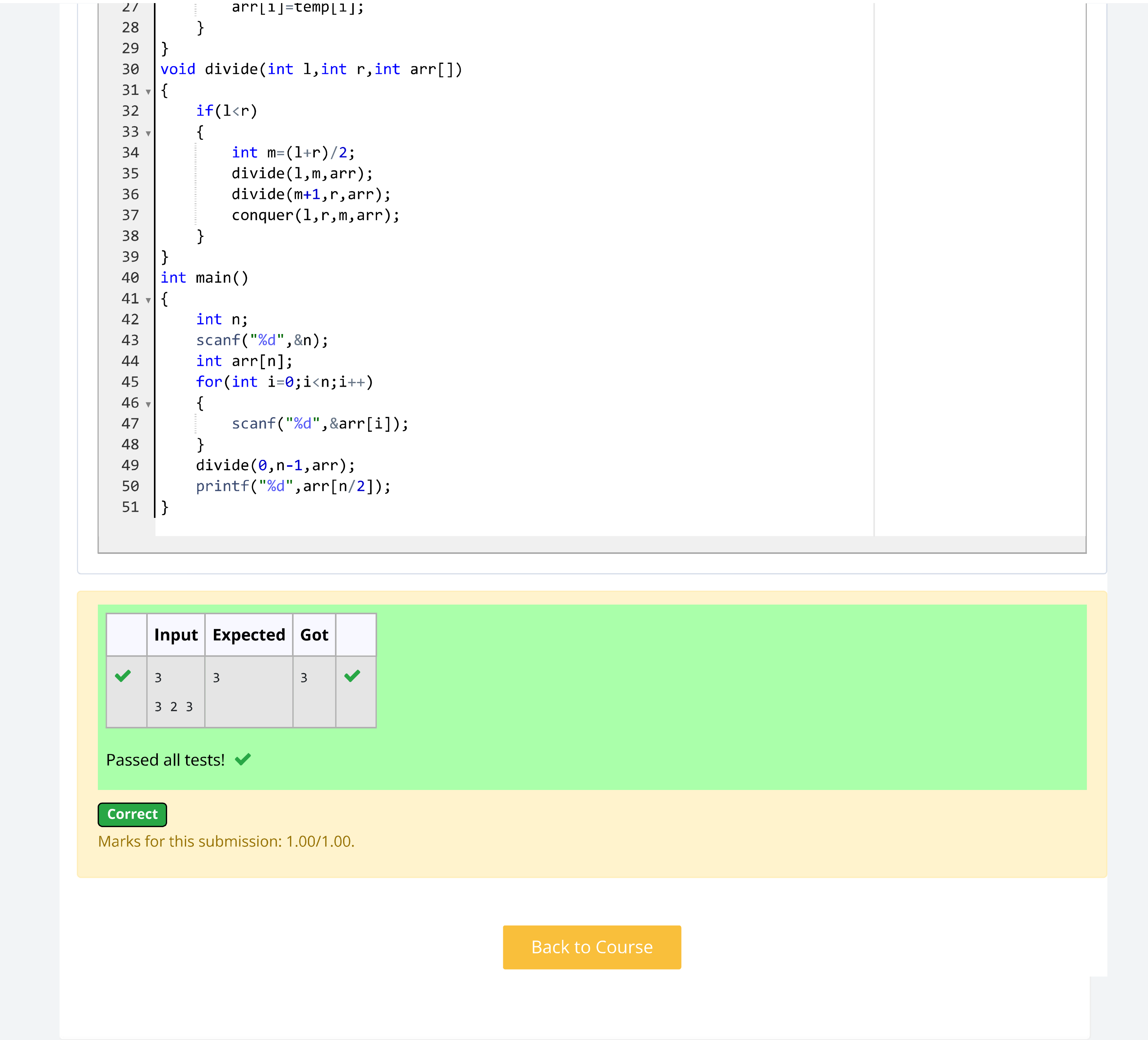
- `n == nums.length`
- `1 <= n <= 5 * ` 10^4
- `-`2^{31}` <= nums[i] <= `2^{31}` - 1`

For example:

Input	Result
3 3 2 3	3
7 2 2 1 1 1 2 2	2

Answer: (penalty regime: 0 %)

```
#include<stdio.h>
void conquer(int l,int r,int mid,int arr[])
{
    int i=l,j=mid,k=0;
    int temp[r-l+1];
    while(i<=mid && j<=r)
    {
        if(arr[i]<=arr[j])
        {
            temp[k++]=arr[i++];
        }
        else
        {
            temp[k++]=arr[j++];
        }
    }
    while(i<=mid)
    {
        temp[k++]=arr[i++];
    }
    while(j<=r)
    {
        temp[k++]=arr[j++];
    }
    for(int i=l;i<=r;i++)
    {
```



```
        arr[i]=temp[i];
    }
}
void divide(int l,int r,int arr[])
{
    if(l<r)
    {
        int m=(l+r)/2;
        divide(l,m,arr);
        divide(m+1,r,arr);
        conquer(l,r,m,arr);
    }
}
int main()
{
    int n;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    divide(0,n-1,arr);
    printf("%d",arr[n/2]);
}
```

	Input	Expected	Got	
✓	3 3 2 3	3	3	✓

Passed all tests! ✓

Marks for this submission: 1.00/1.00.

Question 1 | Correct Mark 1.00 out of 1.00

Problem Statement:

Given a sorted array and a value x, the floor of x is the largest element in array smaller than or equal to x. Write divide and conquer algorithm to find floor of x.

Input Format

First Line Contains Integer n – Size of array

Next n lines Contains n numbers – Elements of an array

Last Line Contains Integer x – Value for x

Output Format

First Line Contains Integer – Floor value for x

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int binarySearch(int l,int r,int arr[],int x)
{
    while(l<=r)
    {
        int mid=(l+r)/2;
        if((arr[mid]<x && arr[mid+1]>x) || (arr[mid]==x))
        {
            return arr[mid];
        }
        else if(arr[mid]>x)
        {
            r=mid-1;
        }
        else
        {
            l=mid+1;
        }
    }
    return 1;
}
int main()
{
    int n,x;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    scanf("%d",&x);
    printf("%d",binarySearch(0,n-1,arr,x));
}
```

	Input	Expected	Got	
✓	6 1 2 8 10 12 19 5	2	2	✓
✓	5 10 22 85 108 129 100	85	85	✓
✓	7 3 5 7 9 11 13 15 10	9	9	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Back to Course

Question 1 | Correct Mark 1.00 out of 1.00

Problem Statement:

Given a sorted array of integers say arr[] and a number x. Write a recursive program using divide and conquer strategy to check if there exist two elements in the array whose sum = x. If there exist such two elements then return the numbers, otherwise print as “No”.

Note: Write a Divide and Conquer Solution

Input Format

First Line Contains Integer n – Size of array

Next n lines Contains n numbers – Elements of an array

Last Line Contains Integer x – Sum Value

Output Format

First Line Contains Integer – Element1

Second Line Contains Integer – Element2 (Element 1 and Elements 2 together sums to value “x”)

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>
#include <stdlib.h>

// Recursive function to find two elements that sum to 'x'
// l: left pointer index, r: right pointer index
int find_sum_recursive(int arr[], int l, int r, int x) {
    // Base Case 1: Pointers have crossed or met (search space is exhausted)
    if (l >= r) {
        return 0; // Return 0 (False)
    }

    int current_sum = arr[l] + arr[r];

    if (current_sum == x) {
        // Base Case 2: Match found
        printf("%d\n", arr[l]);
        printf("%d\n", arr[r]);
        return 1; // Return 1 (True)
    }
    else if (current_sum < x) {
        // The current sum is too small. We need a larger number.
        // Move the left pointer one step to the right to include a larger element.
        // Divide: The problem size is reduced by 1 from the left end.
        return find_sum_recursive(arr, l + 1, r, x);
    }
    else { // current_sum > x
        // The current sum is too large. We need a smaller number.
        // Move the right pointer one step to the left to include a smaller element.
        // Divide: The problem size is reduced by 1 from the right end.
        return find_sum_recursive(arr, l, r - 1, x);
    }
}

int main() {
    int n;
    // Read the size of the array
    if (scanf("%d", &n) != 1) return 1;

    // Use dynamic memory allocation
    int *arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        return 1;
    }

    // Read array elements
    for (int i = 0; i < n; i++) {
        if (scanf("%d", &arr[i]) != 1) {
            free(arr);
            return 1;
        }
    }
```

Question 1 | Correct Mark 1.00 out of 1.00

Problem Statement:

Given a sorted array of integers say arr[] and a number x. Write a recursive program using divide and conquer strategy to check if there exist two elements in the array whose sum = x. If there exist such two elements then return the numbers, otherwise print as “No”.

Note: Write a Divide and Conquer Solution

Input Format

First Line Contains Integer n – Size of array

Next n lines Contains n numbers – Elements of an array

Last Line Contains Integer x – Sum Value

Output Format

First Line Contains Integer – Element1

Second Line Contains Integer – Element2 (Element 1 and Elements 2 together sums to value “x”)

Answer: (penalty regime: 0 %)

```
        // Move the left pointer one step to the right to include a larger element.
        // Divide: The problem size is reduced by 1 from the left end.
        return find_sum_recursive(arr, l + 1, r, x);
    }
    else { // current_sum > x
        // The current sum is too large. We need a smaller number.
        // Move the right pointer one step to the left to include a smaller element.
        // Divide: The problem size is reduced by 1 from the right end.
        return find_sum_recursive(arr, l, r - 1, x);
    }
}

int main() {
    int n;
    // Read the size of the array
    if (scanf("%d", &n) != 1) return 1;

    // Use dynamic memory allocation
    int *arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        return 1;
    }

    // Read array elements
    for (int i = 0; i < n; i++) {
        if (scanf("%d", &arr[i]) != 1) {
            free(arr);
            return 1;
        }
    }

    int x;
    // Read the target sum
    if (scanf("%d", &x) != 1) {
        free(arr);
        return 1;
    }

    // Start the recursive search with pointers at the beginning (0) and end (n-1)
    int found = find_sum_recursive(arr, 0, n - 1, x);

    // If the function returns 0 (False), print "No"
    if (!found) {
        printf("No\n");
    }

    // Clean up memory
    free(arr);

    return 0;
}
```

	Input	Expected	Got	
✓	4 2 4 8 10 14	4 10	4 10	✓
✓	5 2 4 6 8 10 100	No	No	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Back to Course

Question 1 | Correct Mark 1.00 out of 1.00

Write a Program to Implement the Quick Sort Algorithm

Input Format:

The first line contains the no of elements in the list-n

The next n lines contain the elements.

Output:

Sorted list of elements

For example:

Input	Result
5 67 34 12 98 78	12 34 67 78 98

Answer:

```c
#include <stdio.h>
#include <stdlib.h> // Required for malloc and free

// Function to perform the partition (Hoare's scheme)
int quickSort(int arr[], int l, int r)
{
    int pivot = arr[l];
    int i = l - 1;
    int j = r + 1;

    while (1)
    {
        // Find element on the left side that is >= pivot
        do
        {
            i++;
        } while (arr[i] < pivot); // Removed <= to ensure termination when i==j/i==r

        // Find element on the right side that is <= pivot
        do
        {
            j--;
        } while (arr[j] > pivot); // Removed > to ensure termination when j==l/j==i

        // Check for crossing
        if (i >= j)
        {
            // The partition point is j (or i), Hoare's returns j
            return j;
        }

        // Swap the elements
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

// Function to recursively call Quick Sort
void quick(int arr[], int l, int r)
{
    if (l < r)
    {
        // Get the partition point
        int p = quickSort(arr, l, r);

        // Hoare's scheme includes the partition element in the first recursive call
        quick(arr, l, p);
        quick(arr, p + 1, r);
    }
}

```

Question 1 | Correct Mark 1.00 out of 1.00

Write a Program to Implement the Quick Sort Algorithm

Input Format:

The first line contains the no of elements in the list-n

The next n lines contain the elements.

Output:

Sorted list of elements

For example:

Input	Result
5 67 34 12 98 78	12 34 67 78 98

Answer:

```

// Function to recursively call Quick Sort
void quick(int arr[], int l, int r)
{
    if (l < r)
    {
        // Get the partition point
        int p = quickSort(arr, l, r);

        // Hoare's scheme includes the partition element in the first recursive call
        quick(arr, l, p);
        quick(arr, p + 1, r);
    }
}

int main()
{
    int n;
    // Read size
    if (scanf("%d", &n) != 1) return 1;

    // FIX 3: Use Dynamic Memory Allocation
    int *arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        return 1;
    }

    // Read elements
    for (int i = 0; i < n; i++)
    {
        if (scanf("%d", &arr[i]) != 1) {
            free(arr);
            return 1;
        }
    }

    // Perform Quick Sort
    quick(arr, 0, n - 1);

    // Print sorted array
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Clean up memory
    free(arr);

    return 0;
}
```

	Input	Expected	Got	
✓	5 67 34 12 98 78	12 34 67 78 98	12 34 67 78 98	✓
✓	10 1 56 78 90 32 56 11 10 90 114	1 10 11 32 56 56 78 90 90 114	1 10 11 32 56 56 78 90 90 114	✓
✓	12 9 8 7 6 5 4 3 2 1 10 11 90	1 2 3 4 5 6 7 8 9 10 11 90	1 2 3 4 5 6 7 8 9 10 11 90	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Back to Course