Ex. No: 4 **Date: 03.09.24**

Register No.: 230701368 **Name: AL UMA**

DIVIDE AND CONQUER

4.A Number Of Zeros In A Given Array

AIM:

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.

ALGORITHM:

Function index(a[], l, r)

// Step 1: Calculate the mid-point of the current range

Initialize mid as l + (r - l) / 2

// Step 2: Base cases to return index

If a[0] is 0

Return 0 // The first element is 0, return index 0

Else If a[r-1] is 1

Return r // The last element is 1, return index r

```
    // Step 3: Recursively search for the index of 1s
    If a[mid] is 0 and a[mid-1] is 0
        // Recursively search in the left half
        Return index(a, l, mid)
    Else If a[mid] is 0
        // Return the current mid index, since it's the transition point
        Return mid
    Else
        // Recursively search in the right half
        Return index(a, mid+1, r)

End Function



Function main()
    // Step 1: Read the number of elements n
    Initialize n  // Number of elements
    Read n from user  // Get the input value for n

    // Step 2: Declare the array and read values
    Initialize array a of size n  // Declare the array of size n
    For i from 0 to n-1  // Loop to input values into array a
        Read a[i] from user  // Input value into array a[i]
    End For

    // Step 3: Call the index function to find the first index of 1
    Initialize c as index(a, 0, n)  // Call the index function to find the correct index

    // Step 4: Print the result
```

```
    Print n - c  // Output the count of 1s, which is n - c
End Function
```

PROGRAM:

```
#include<stdio.h>
int index(int a[],int l,int r)
{
    int mid=0;
    mid=l+(r-l)/2;
    if (a[0]==0)
        return 0;
    else if (a[r-1]==1)
        return r;
    if ((a[mid]==0) && (a[mid-1]==0))
        return index(a,0,mid);
    else if (a[mid]==0)
        return mid;
    else
        return index(a,mid+1,r);


}



int main()
{
int n;
```

```
scanf("%d",&n);
int a[n];
for(int i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
int c=index(a,0,n);
printf("%d",n-c);

}
```

OUTPUT:

	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	✔

4.B 2-Majority Elements

AIM:

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

ALGORITHM:

Function count(a[], l, r, k)

// Step 1: Base case, exit if the current range is invalid

If l >= r

Return 0 // Return 0 if the range is invalid

// Step 2: Initialize mid as the middle index of the range

Initialize mid as (l + r - 1) / 2

```
    // Step 3: Check if the element at mid equals k and count it
    If a[mid] is equal to k
        Increment global counter c  // Increment count if a[mid] is k

    // Step 4: Recursively search the left half
    Call count(a, l, mid, k)

    // Step 5: Recursively search the right half
    Call count(a, mid + 1, r, k)

    // Step 6: Return the final count of occurrences of k
    Return c
End Function


Function main()
    // Step 1: Read the number of elements n
    Initialize n  // Number of elements
    Read n from user  // Get the input value for n

    // Step 2: Declare the array and read its values
    Initialize array a of size n  // Declare the array of size n
    For i from 0 to n-1  // Loop to input values into array a
        Read a[i] from user  // Input value into array a[i]
    End For

    // Step 3: Get the value of k (the first element of the array)
    Initialize k as a[0]  // Set k to the first element of the array

    // Step 4: Count the occurrences of k in the array using the count function
```

```
    If count(a, 0, n, k) > n / 2
        // Step 4.1: If k appears more than n/2 times, print k
        Print k
    Else
        // Step 4.2: Otherwise, find the first element not equal to k
        For i from 0 to n / 2  // Loop to check the first half of the array
            If a[i] is not equal to k
                // Step 4.3: Print the first element that is not equal to k
                Print a[i]
                Break  // Exit the loop after finding the first mismatch
        End For
    End If
End Function
```

PROGRAM:

```
#include<stdio.h>
int c=0;
int count(int a[],int l,int r,int k)
{
if(l<r)
{
int mid =(l+r-1)/2;
if(a[mid]==k)
{
c++;
}
count(a,l,mid,k);
count(a,mid+1,r,k);
}
```

```
return c;
}
int main()
{
    int n;
    scanf("%d",&n);
    int a[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    int k=a[0];



    if(count(a,0,n,k)>n/2)
    printf("%d",k);

    else
    {
        for(int i=0;i<n/2;i++)
        {
            if(a[i]!=k)
            {
                printf("%d",a[i]);
                break;

            }
        }
    }
}
```

OUTPUT:

	Input	Expected	Got	
✔	3 3 2 3	3	3	✔

4.C Finding Floor Values

AIM:

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

ALGORITHM

Function maxfloor(a[], l, r, d)

// Step 1: Base case, exit if the current range is invalid

If l >= r

Return h // Return the current maximum value of h

// Step 2: Calculate the mid-point of the current range

Initialize mid as (l + r - 1) / 2

// Step 3: Check if a[mid] divided by d is 0 and a[mid] is greater than h

If a[mid] divided by d equals 0 and a[mid] is greater than h

// Update h to the value of a[mid] if the condition is met

Set h as a[mid]

// Step 4: Recursively search the left half

Call maxfloor(a, l, mid, d)

```
    // Step 5: Recursively search the right half
    Call maxfloor(a, mid + 1, r, d)

    // Step 6: Return the maximum value h found during the search
    Return h
End Function



Function main()
    // Step 1: Read the number of elements n
    Initialize n  // Number of elements
    Read n from user  // Get the input value for n

    // Step 2: Declare the array and read its values
    Initialize array a of size n  // Declare the array of size n
    For i from 0 to n-1  // Loop to input values into array a
        Read a[i] from user  // Input value into array a[i]
    End For

    // Step 3: Read the value of d
    Initialize d  // Declare d (used for checking a[mid] divided by d)
    Read d from user  // Input the value of d

    // Step 4: Call maxfloor to find the maximum value of h
    Initialize result as maxfloor(a, 0, n, d)  // Call the maxfloor function to
calculate h

    // Step 5: Print the result
    Print result  // Output the result (maximum value of h)
End Function
```

PROGRAM:

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

OUTPUT

	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	✔

4.D 4-Two Elements Sum To X

AIM:.

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")

ALGORITHM:

Function sum(a[], l, r, s)

// Step 1: Base case, exit if the range is invalid (l >= r)

If l >= r

Return 0 // No valid pair found, return 0

// Step 2: Calculate mid-point

Initialize mid as (l + r) / 2

// Step 3: Check if the sum of a[mid] and a[r] equals s

If a[mid] + a[r] equals s

// Step 3.1: If the sum equals s, store the values and return 1

Set s1 as a[mid] // Store a[mid] in s1

Set s2 as a[r] // Store a[r] in s2

Return 1 // Pair found, return 1

// Step 4: Recursively search by reducing the right index

```
    Call sum(a, l, r - 1, s)  // Recursively check the subarray a[l, r-1]

    Return 0  // No valid pair found in the current range
End Function


Function main()
    // Step 1: Read the number of elements n
    Initialize n  // Number of elements in the array
    Read n from user  // Get the input value for n

    // Step 2: Declare the array a of size n and read its values
    Initialize array a of size n  // Declare an array of size n
    For i from 0 to n-1  // Loop to input values into array a
        Read a[i] from user  // Input value into array a[i]
    End For

    // Step 3: Read the value of x (the sum target)
    Initialize x  // Declare the target sum x
    Read x from user  // Input the value of x

    // Step 4: Call sum function to check for a pair whose sum equals x
    Initialize y as sum(a, 0, n-1, x)  // Call the sum function

    // Step 5: Output the result based on the return value of sum
    If y equals 0  // If no pair is found
        Print "No"  // Output "No"
    Else  // If a valid pair is found
        Print s1  // Output the first element of the pair
        Print s2  // Output the second element of the pair
```

End Function

PROGRAM

```
#include<stdio.h>
int s1=0,s2=0;
int sum(int a[],int l,int r,int s)
{
    if(l<r)
    {
        int mid=(l+r)/2;
        if(a[mid]+a[r]==s)
        {
            s1=a[mid];
            s2=a[r];
            return 1;
        }
        sum(a,l,r-1,s);

    }
    return 0;

}
int main()
{
    int n;
    scanf("%d",&n);
    int a[n];
    for(int i=0;i<n;i++)
    {
```

```
        scanf("%d",&a[i]);
    }
    int x;
    scanf("%d",&x);
    int y=sum(a,0,n-1,x);
    if (y==0)
    printf("%s","No");
    else
    {
        printf("%d\n%d",s1,s2);
    }

}
```

OUTPUT:

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

4.E Implement Quick Sort

AIM:

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

.

ALGORITHM:

Function quick(a[], l, r)

// Step 1: Base case, exit if the range is invalid

If l >= r

Return // No sorting needed for invalid range

// Step 2: Set the pivot as the middle element of the range

Initialize pivot as (l + r) / 2

// Step 3: Initialize two pointers i and j

Initialize i as l // Pointer i starts from the left

Initialize j as r // Pointer j starts from the right

// Step 4: Partition the array based on the pivot

While i < j // Continue partitioning while the pointers do not cross

// Step 4.1: Increment i until a[i] is greater than pivot

While a[i] <= a[pivot] // Move i to the right as long as a[i] <= pivot

```
        Increment i  // Increase i to find an element greater than pivot

    // Step 4.2: Decrement j until a[j] is smaller than pivot
    While a[j] > a[pivot]  // Move j to the left as long as a[j] > pivot
        Decrement j  // Decrease j to find an element smaller than pivot

    // Step 4.3: If pointers i and j are valid (i <= j), swap elements
    If i <= j
        Swap a[i] with a[j]  // Swap the elements at indices i and j

  End While

  // Step 5: Swap the pivot with a[j] to place it in the correct position
  Swap a[j] with a[pivot]  // Place the pivot element at its correct sorted position

  // Step 6: Recursively sort the left and right subarrays
  Call quick(a, l, j-1)  // Recursively sort the left subarray from l to j-1
  Call quick(a, j+1, r)  // Recursively sort the right subarray from j+1 to r
End Function

Function main()
  // Step 1: Read the number of elements n
  Initialize n  // Number of elements in the array
  Read n from user  // Get the input value for n

  // Step 2: Declare the array a of size n and read its values
  Initialize array a of size n  // Declare an array of size n
  For i from 0 to n-1  // Loop to input values into array a
    Read a[i] from user  // Input value into array a[i]
  End For
```

```
    // Step 3: Call quick function to sort the array
    Call quick(a, 0, n-1)  // Call quick sort on the array a from 0 to n-1

    // Step 4: Print the sorted array
    For i from 0 to n-1  // Loop to print each element of the sorted array
        Print a[i] followed by a space  // Output the sorted element
    End For
End Function
```

PROGRAM

```
#include<stdio.h>
void quick(int a[],int l,int r)
{
    if(l<r)
    {

        int pivot=(l+r)/2;
        int i=l;

        int j=r;

        while(i<j)
        {

            while(a[pivot]>=a[i] )
            {
                i++;
```

```
        }

        while(a[pivot]<a[j] )
        {
            j--;
        }
        if(i<=j)
        {
            int temp=a[i];
            a[i]=a[j];
            a[j]=temp;
        }

    }
    int temp=a[j];
    a[j]=a[pivot];
    a[pivot]=temp;
    quick(a,l+1,r);



}
}
int main()
{
    int n;
    scanf("%d",&n);
    int a[n];
    for(int i=0;i<n;i++)
    {
```

```
        scanf("%d",&a[i]);
    }
    quick(a,0,n-1);
    for(int i=0;i<n;i++)
    {
        printf("%d ",a[i]);
    }
}
```

OUTPUT:

	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	✔