NAME : HARISH R P

REG.NO : 230701106

DEPT : B E COMPUTER SCIENCE AND ENGINEERING - A

Divide and Conquer

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 count(a, left, right) {

// base case: if left index exceeds right index if left is greater than right {

return 0

}

initialize mid as (left + right) / 2 // find the middle index

```
// check if the middle element is 1 if a[mid] is equal to 1 {

// check if the next element is 0 if a[mid + 1] is equal to 0 {

// count zeros from mid + 1 to right


initialize c as (right - (mid + 1)) + 1 return c

} else {


// search in the right half return count(a, mid + 1, right)

}

}

// check if both ends are 0

else if a[left] is equal to 0 and a[right] is equal to 0 { return right + 1 // return total count of elements

}

// search in the left half else {

return count(a, left, mid - 1)

}

}




function main() {

initialize n // number of elements read n from user


initialize arr array of size n // array to hold binary values
```

// read values into the arr array for i from 0 to n - 1 {

read arr[i] from user

}

initialize left as 0 // left index initialize right as n - 1 // ri

Program:

```
#include <stdio.h>
int count(int a[],int left,int right)
{
if(left>right)
{
return 0;
}
int mid=(left+right)/2; if(a[mid]==1)
{
if(a[mid+1]==0)
{
int c= (right-(mid+1))+1; return c;
}
else{
return count(a,mid+1,right);
}
}
else if(a[left]==0 && a[right]==0)
```

```
{
return right+1;
}
else

{
return  count(a,left,mid-1);
}



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

int left=0; int right=n-1;
int result=count(arr,left,right); printf("%d",result);
}
```

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	✔

a. **Majority Element**

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:

int divide(a, l, r, n) {

// base case: if left index equals right index if l is equal to r {

return a[l] // return the only element

}

initialize mid as (l + r) / 2 // find the middle index

// recursively divide the array

initialize min as divide(a, l, mid, n) // find min in left half initialize max as divide(a, mid + 1, r, n) // find max in right half

initialize leftc as 0 // counter for min occurrences initialize rightc as 0 // counter for max occurrences

// count occurrences of min and max in the entire array for i from 0 to n - 1 {

if a[i] is equal to min {

increment leftc by 1 // count occurrences of min

} else {

increment rightc by 1 // count occurrences of max

}

}

// check if min occurs more than n/2 times if leftc is greater than (n / 2) {

return min // return min if it is the majority element

} else {

return max // return max otherwise

}

}

int main() {

initialize n // number of elements read n from user

initialize a array of size n // array to hold input values

// read values into the array for j from 0 to n - 1 {

read a[j] from user

}

initialize l as 0 // left index initialize r as n - 1 // right index

// call the divide function initialize result as divide(a, l, r, n)

print result // output the final majority element

}

Program:

```
#include<stdio.h>

int divide(int a[],int l,int r,int n){ if(l==r)

{

return a[l];

}

int mid=(l+r)/2;

int min=divide(a,l,mid,n); int max=divide(a,mid+1,r,n); int leftc=0,rightc=0; for(int i=0;i<n;i++)

{

if(a[i]==min)

{

leftc++;

}

else
```

```
{
rightc++;
}
}
if(leftc>(n/2))
{
return min;



}
else
{
return max;
}
}
int main(){ int n;
scanf("%d",&n); int a[n];
for(int j=0;j<n;j++){ scanf("%d",&a[j
]);
}
int l=0,r=n-1;
int result=divide(a,l,r,n); printf("%d",result);
}
```

Output:

	Input	Expected	Got	
✔	3 3 2 3	3	3	✔

a. Finding Floor Value

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:

int large(arr, l, r, x){

// Base case: if the range is invalid if r < l

return 0 // return 0 when there is no valid element

// Calculate the middle index mid = (l + r) / 2

// Check if the middle element is equal to x if arr[mid] is equal to x

return mid // return the index of x if found

```
// If the middle element is less than x else if arr[mid] < x

// Recursively search in the right half floorIndex = large(arr, mid + 1, r, x)


// Check if a valid floor index is found


if floorIndex is not equal to 0

return floorIndex // return the found index else

return mid // return mid as the largest element less than x




// If the middle element is greater than x, search in the left half else

return large(arr, l, mid - 1, x) // search in the left half


}

Int main()

initialize n // number of elements in the array read n from user


initialize arr of size n // array to hold input values




// Read values into the array for i from 0 to n - 1

read arr[i] from user




initialize l as 0 // left index initialize r as n - 1 // right index
```

initialize x // the value for which we want to find the largest element less than or equal to

x

read x from user

// Call the large function result = large(arr, l, r, x)

// Check the result if result is equal to 0

print x // if no valid element, print x else

print arr[result] // print the largest element less than or equal to x

Program:

```
#include<stdio.h>

int large(int arr[],int l,int r,int x){ if (r < l) {

return 0;

}

int mid=(l+r)/2; if (arr[mid]==x)

{

return mid;

}

else if (arr[mid]<x)

{

int floorIndex=large(arr,mid+1,r,x); if(floorIndex!=0)

{
```

```
return floorIndex;
}
else
{
return floorIndex=mid;

}
}
else
{
return large(arr,l,mid-1,x);
}
}


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

int l=0; int
r=n-1; int x;
scanf("%d",&x); int
result=large(arr,l,r,x); if
(result == 0)
{
```

```
printf( "%d",x);

}

else

{

printf( "%d",arr[result]);

}


}
```

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	✔

a. 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:

int findPairWithSum(arr, left, right, x){

// Base case: if there are no more pairs to check if left >= right

print "No" // No pair found return

// Calculate the sum of the elements at the left and right indices sum = arr[left] + arr[right]

// Check if the sum is equal to x if sum is equal to x

print arr[left] // Print the first element of the pair print arr[right] // Print the second element of the pair return

// If the sum is less than x, move the left index up

if sum < x

findPairWithSum(arr, left + 1, right, x) // Recursive call with increased left index else findPairWithSum(arr, left, right - 1, x) // Recursive call with decreased right index

}

function main()

initialize n // number of elements in the array read n from user

initialize arr of size n // array to hold input values

// Read values into the array for i from 0 to n - 1

read arr[i] from user

initialize x // the target sum value read x from user

// Call the findPairWithSum function findPairWithSum(arr, 0, n - 1, x)

Program:

#include <stdio.h>

void findPairWithSum(int arr[], int left, int right, int x) { if (left >= right) {

//No pair found

printf("No\n")

; return;

}

int sum = arr[left] + arr[right];

if (sum == x){

// If the pair is found printf("%d\n%d\n", arr[left], arr[right]); return;

```
}

if (sum < x){

findPairWithSum(arr, left + 1, right, x);
}
else{

findPairWithSum(arr, left, right - 1, x);
}
}
int main()
{ int n; scanf("%d",
&n); int arr[n];
for (int i = 0; i < n; i++) { scanf("%d", &arr[i]);
}

int x; scanf("%d", &x);
findPairWithSum(arr, 0, n - 1, x);

}
```

Output:

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

a. Implementation of 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:

int partition(a, left, right)

{

pivot = right // Choose the last element as pivot i

= left - 1 // Index of smaller element

```
for j from left to right - 1

{

if a[j] < a[pivot]

{

i++

// Swap a[i] and a[j] temp = a[i]

a[i] = a[j] a[j] = temp

}


}




// Swap a[i + 1] and a[right] temp = a[i + 1] a[i + 1] = a[right]


a[right] = temp

return (i + 1) // Return the partition index

}




function quick(a, left, right)

{

if left < right

{

p = partition(a, left, right) // Partition the array

quick(a, left, p - 1)// Recursively sort the left sub-array quick(a, p + 1, right)// Recursively sort the right
```

```
sub-array
}


}



int main()
{
initialize n // number of elements read n from user


initialize a of size n // array to hold input values for i from 0 to n - 1
{
read a[i] from user
}



quick(a, 0, n - 1) // Call the quicksort function



// Print the sorted array


for i from 0 to n - 1
{
print a[i]
}
}
```

Program:

```
#include <stdio.h>

int partition(int a[], int left, int right) { int pivot = right;
int i = left-1;

for (int j = left; j < right; j++) {

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

int temp = a[i + 1]; a[i + 1] =
a[right]; a[right]
= temp; return (i
+ 1);

}
```

```
void quick(int a[], int left, int right)
{ if (left < right) {
int p = partition(a, left, right); quick(a, left, p - 1); quick(a, p + 1, right);
}
}


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	✔