Aim:

To implement Min-Max and Binary Search in an array using Divide & Conquer Approach.

Theory:

Min-Max: The divide and conquer approach is an efficient way to find the minimum and maximum values in an array. The algorithm works by dividing the array into two halves, recursively finding the minimum and maximum values in each half, and then combining the results to obtain the minimum and maximum values for the entire array. The time complexity of the divide and conquer algorithm for finding the minimum and maximum values in an array is O(n log n), which is much faster than the simple linear search algorithm that has a time complexity of O(n^2).The idea is to recursively divide the array into two equal parts and update the maximum and minimum of the whole array in recursion by passing minimum and maximum variables by reference. The base conditions for the recursion will be when the subarray is of length 1 or 2. It is an important tool for many practical applications and is a fundamental concept in computer science and algorithm design.

Divide: Divide array into two halves.

Conquer: Recursively find maximum and minimum of both halves.

Combine: Compare maximum of both halves to get overall maximum and compare minimum of both halves to get overall minimum.

Binary Search:

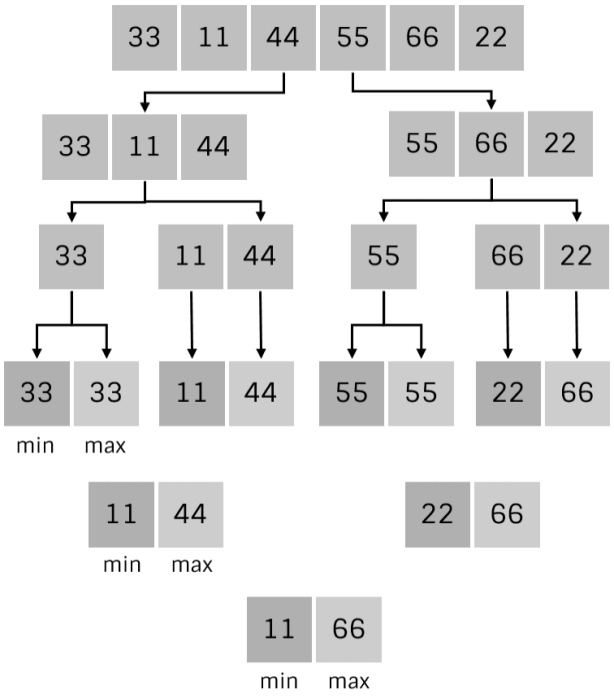
Binary search using the divide and conquer algorithm is an efficient and widely used algorithm for finding a specific element in a sorted array. The algorithm works by repeatedly dividing the search interval in half until the target element is found or it is determined that the element is not in the array. The divide and conquer approach reduces the search interval by half at each step, resulting in a time complexity of O(log n), where n is the size of the input array.

In practice, binary search is used in a wide range of applications, such as searching a database, searching for a word in a dictionary, or finding a specific item in a sorted list of items. The divide and conquer algorithm provides a simple and elegant way to perform the search, and its time complexity makes it an attractive option for large datasets.

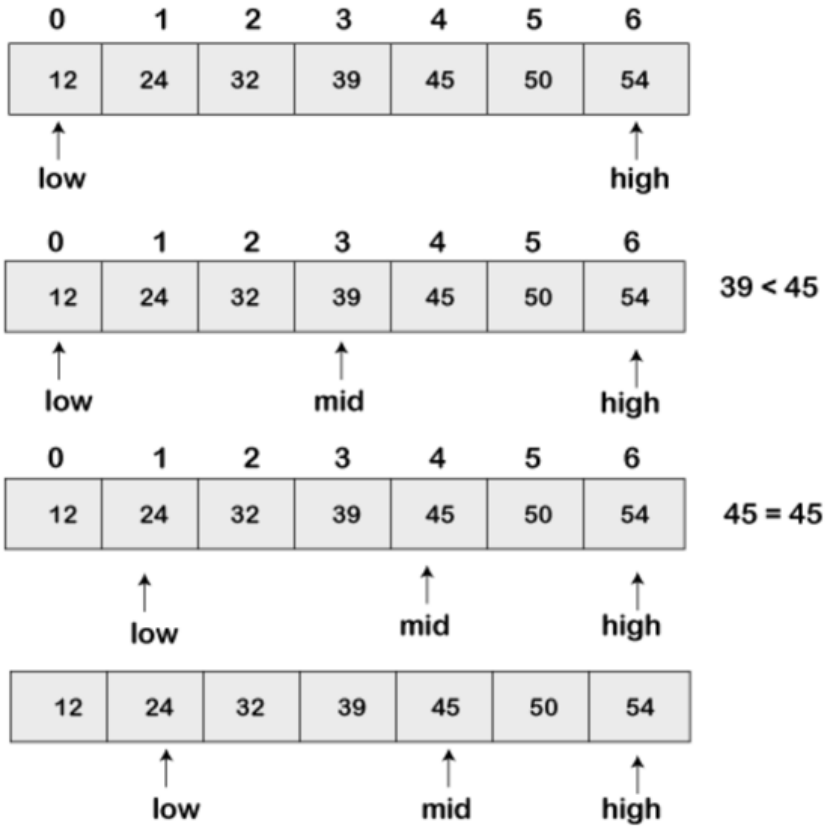
However, it is important to note that binary search only works on sorted arrays. If the array is not sorted, the algorithm will not work correctly. Additionally, if the target element appears multiple times in the array, the algorithm may return any of the indices where the element appears, not necessarily the first or last occurrence. Overall, binary search using the divide and conquer algorithm is a powerful tool for searching sorted arrays and has numerous practical applications.

Example:

Min-Max:



Binary Search:



Algorithm:

Min_Max(A,i,j,max,min)

```
If(i==j)
        max=min=A[i]
else if(i=j-1)
        if A[i]<A[j]
                min=A[i]
                max=A[j]
        else
                max=A[i]
                min=A[j]
else
        mid=(i+j)/2
        Min_Max(A,i,mid,max,min)
        Min_Max(A,mid+1,j,max1,min1)
        If max1>max
                max=max1
        If min1<min
                min=min1
```

Binary_Search(A,low,high,X)

```
If(low==high)
        If(A[low]==X)
                Return low
        Else
                Return -1
Else
        Mid=(low+high)/2
        If(A[mid]==X)
                Return mid
        Else if(A[mid]>X)
                Binary_Search(A,low,mid-1,X)
        Else
                Binary_Search(A,mid+1,high,X)
```

Code:

Min-Max:

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

int max, min;
int a[100];

void maxmin(int i, int j) {
   int max1, min1, mid;
   if(i == j) {
      max = min = a[i];
   }
   else {
      if(i == j-1) {
         if(a[i] < a[j]) {
            max = a[j];
            min = a[i];
         }
         else {
            max = a[i];
            min = a[j];
         }
      }
      else {
         mid = (i+j)/2;
         maxmin(i, mid);
         max1 = max;
         min1 = min;
```

```
            maxmin(mid+1, j);
            if(max < max1)
               max = max1;
            if(min > min1)
               min = min1;
         }
      }
   }

int main () {
   int i, num;
   clock_t start, end;
   double cpu_time_used;
   start = clock();

   printf("\nEnter the total number of numbers : ");
   scanf("%d", &num);
   printf("Randomly generated array of size %d : \n", num);
   srand(time(NULL));
   for(i = 0; i < num; i++) {
      a[i] = rand() % 100 + 1;
      // printf("%d ", a[i]);
   }

   max = a[0];
   min = a[0];
   maxmin(0, num-1);
   printf("\nMinimum element in the array : %d\n", min);
   printf("Maximum element in the array : %d\n", max);
   end = clock();
```

```
    cpu_time_used = (double)(end-start);
    printf("\ntime taken: %f",cpu_time_used);
    return 0;
}
```

Binary Search:

```
#include<stdio.h>
#include<conio.h>
#include<time.h>

int binarySearch(int arr[], int x, int low, int high)
{int mid;
        if (low > high)
            return -1;

        else
            mid = (low + high) / 2 ;
            if (arr[mid] == x)
                return mid;

            else if (x > arr[mid])
                return binarySearch(arr, x, mid + 1, high);

            else
                return binarySearch(arr, x, low, mid - 1) ;
}
void main(){
    int a[10000], n,i,x,ans;
    clock_t start, end;
```

```
    double cpu_time_used;
    start = clock();
    printf("\nenter number of elements: ");
    scanf("%d",&n);
    //printf("\nelements: ");
    for(i=0;i<n;i++){
    // scanf("%d",&a[i]);
    a[i] = i;
    }
    printf("\nenter element to be searched: ");
    scanf("%d",&x);
    ans = binarySearch(a, x, 0, n-1);
    if(ans == -1)
        printf("\nnot found");
    else
        printf("\n%d found at %d",x,ans);
        end = clock();
        cpu_time_used = (double)(end-start);
        printf("\ntime taken: %f",cpu_time_used);
}
```

Output:

Min-Max:

```
Enter the total number of numbers : 1000
Randomly generated array of size 1000 :

Minimum element in the array : 1
Maximum element in the array : 100

time taken: 130.000000

...Program finished with exit code 0
Press ENTER to exit console.
```

Binary Search:

```
enter number of elements: 1000

enter element to be searched: 500

500 found at 500
time taken: 119.000000

...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

The Divide and Conquer Approach is used to easily calculate Min-Max in an array or perform Binary Search.