Practical No. - 01

Aim - Implementation of Binary search using Divide and conquer method.

Theory - Binary search is an algorithm that works by repeatedly dividing the search interval in half until the target value is found or determined not tobe present in the array. This algorithm is based on the principle of divide and conquer and is highly efficient for large sorted arrays.

The time complexity of binary search is O(logn), where n is the size of input array. However, binary search requires an array tobe sorted, which can add on extra O(nlogn) complexity if the array is unsorted.

Binary search is typically used for large sorted arrays, where it is highly efficient and can provide significant performance gain over linear search.

```
Function binary-search (A, n, I)
is
    L := 0
    R := n - 1
while L ≤ R do
    m := floor ((L+R)/2)
    if A[m] < I then
        L := m+1
    else if A[m] > I then
        R := m-1
    else:
        return m
    return unsuccessful
```

BINARY SEARCH:

CODE:

```c
#include<stdio.h>
int main()
{
    int a[100],i,flag=0,lb,ub,mid,pos,item,n;

    printf("Enter the range:");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        printf("Enter the element %d:",i+1);
        scanf("%d",&a[i]);
    }
    printf("Enter the item you want to search:");
    scanf("%d",&item);

    lb=0;
    ub=n-1;

    while(lb<=ub)
    {
        mid=(lb+ub)/2;
        if(item==a[mid])
        {
            flag=1;
            pos=mid;
            break;
        }
        else if(item>a[mid])
        lb=mid+1;
        else
        ub=mid-1;
    }
    if(flag==1)
    printf("Item found at position %d",mid+1);
    else
    printf("Not found");
return 0;
}
```

OUTPUT:

```
Enter the range:5
Enter the element 1:1
Enter the element 2:2
Enter the element 3:3
Enter the element 4:4
Enter the element 5:5
Enter the item you want to search:3
Item found at position 3
--------------------------------
Process exited after 11.54 seconds with return value 0
Press any key to continue . . .
```

Time complexity :

Best	O(1)
Average	O(log n)
Worst	O(log n)

space complexity : O(1)

Result - We have successfully implemented Binary search algorithm.

Practical - 02

Aim - Implementation of merge sort and quick sort using Divide and conquer method. Determine the time required to sort the elements.

Theory - Merge sort is a popular comparison based sorting algorithm that follows the divide and conquer strategy to sort an array or a list. It breaks the input array into smaller subarrays, recursively sorts them, and then merges the sorted sub arrays to produce the final sorted array.
The key steps are as follows:

1) Divide: The input array is divided into two equal or nearly equal halves.
2) Conquer: The two halves are sorted recursively using merge sort algorithm.
3) Merge: The sorted halves are merged to produce the final sorted array.

Algorithm:

```
func merge (A, B)
  c = new Array ()
  while (A has elem & B has elem)
  If A[i] > B[i]
      (.addB[i]);
       B++
  else : (.add (A[i])
```

return c

```
func merge_sort (A)
    m = middle of A.
  B = merge_sort (A[0:middle])
  c = merge_sort (A[middle+1:-1])
return merge (B, C)
```

Time complexity :
Best-case - $\Omega(n \log n)$
Average-case - $\Theta(n \log n)$
worst - $O(n \log n)$

space complexity: $O(n)$

Quick sort : Quick sort is a widely used sorting algorithm known for its efficency and the effectiveness in sorting large datasets.

Algorithm:

```
func quicksort (A, Lo, hi)
  if Lo ≥ 0 && hi ≥ 0 && Lo < hi then
      p := partition (A, Lo, hi)
      Quicksort (A, p + 1, hi)
  func partition (A, Lo, hi)
      pivot := A [Floor (hi - Lo) /2) + Lo]
       i := lo - 1
       j = hi + 1
```