**Experiment-1**

**Aim**

Implement Recursive and Iterative Binary search and determine the time taken to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

**Theory**

**Binary search** is an efficient algorithm used to find an element in a sorted list by repeatedly dividing the search interval in half, achieving a time complexity of O(logn). Both **recursive** and **iterative** implementations have similar performance, but the iterative approach is slightly faster and more memory-efficient due to the absence of recursive call overhead.

**Software Used –** Visual Studio Code

**Code-**

**Iterative**

#include<iostream>

using namespace std;

int main() {

// code here

cout<<"Enter the size of the array";

int n; cin>>n;

int arr[n];

cout<<"Enter the elements of the array";

for(int i=0;i<n;i++) cin>>arr[i];

cout<<"Enter the Key to be Found";

int key; cin>>key;

int low=0;

int high =n-1;

int flag=0;

while(low<=high){

int mid=low + (high-low)/2;

if(arr[mid]==key){

cout<<"The element is at position "<< mid+1<<endl;

flag=1;

break;

}else if(arr[mid]>key){

high=mid-1;

}else{

low=mid+1;

}

}

if(flag==0) cout<<"Element Not Found"<<endl;

return 0;

}

**Recursive-**

#include<iostream>

using namespace std;

int search (int arr[],int k, int low,int high){

if(low==high){

if(arr[low]==arr[high]) return low;

else return -1;

}

int mid=low+(high-low)/2;

if(arr[mid]==k) return mid;

else if(arr[mid]>k) return search(arr,k,low,mid-1);

else return search(arr,k,mid+1,high);

}

int main() {

// code here

cout<<"Enter the size of the array";

int n; cin>>n;

int arr[n];

cout<<"Enter the elements of the array";

for(int i=0;i<n;i++) cin>>arr[i];

cout<<"Enter the Key to be Found";

int k; cin>>k;

int pos=search(arr,k,0,n-1);

if(pos==-1){

cout<<"element not found";

}else{

cout<<"element is found at position "<<pos+1;

}

return 0;

}

**Output-**





