**Experiment No:** 1  **Date:** 30/03/2021

**Aim:** Iterative and Recursive Binary Search Algorithm Using

Divide and Conquer & Estimating Its Step Count

**Theory:**

**Binary Search:**

* Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval overing the whole array.
* If the value of the search key is less than the item in the middle of the interval, arrow the interval to the lower half.
* Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.
* We basically ignore half of the elements just after one comparison.
  + Compare x with the middle element.
  + If x matches with middle element, we return the mid index.
  + Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So, we recur for right half.
  + Else (x is smaller) recur for the left half.

**Iteration Algorithm:**

* In Iteration algorithm, certain set of statements are repeated a certain number of times.
* An Iterative algorithm will use looping statements such as for loop, while loop or do-while loop to repeat the same steps number of times.

**Recursion Algorithm:**

* Recursive algorithm, a function calls itself again and again till the base condition(stopping condition) is satisfied.

**Iterative Binary Search**

* The main() method of Iterative Binary Search class starts off with defining an Array of size 6, named A.
* Key is the number to be searched in the list of elements.
* Inside the while loop, "mid" is obtained by calculating (low+high)/2.
* If number at position mid equal to key or target element then the control returns index of mid element by printing that the number has been found along with the index at which it was found.
* Else, if key or target is less than number at position mid then the portion of the Array from the mid upwards is removed from contention by making "high" equal to mid-1.
* Else, it implies that key element is greater than number at position mid(as it is not less than and also not equal, hence, it has to be greater). Hence, the portion of the list from mid and downwards is removed from contention by making "low" equal to mid+1.
* The while loop continues to iterate in this way till either the element is returned (indicating key has been found in the Array) or low becomes greater than high, in which case -1 is returned indicating that key could not be found and the same is printed as output.

**ALGORITHM:**

Algorithm BinSrch ( a , n , x )

//Given an array a[1: n] of elements in non-decreasing order,

//n≥0, determine whether x is present, and

//if so, return j such that x = a[j]; else return 0.

{

low := 1; high := n;

while ( low ≤ high ) do

{

mid :=[( low + high )/2] ;

if ( x < a[i] ) then high := mid-1;

else if ( x > a[mid] ) then low := mid+1;

else return mid;

}

return 0;

}

**Recursive Binary Search**

* Recursive implementation of binary search algorithm, in the method binarySearch(), follows almost the same logic as iterative version, except for a couple of differences.
* The first difference is that the while loop is replaced by a recursive call back to the same method with the new values of low and high passed to the next recursive invocation along with "Array" and "key" or target element.
* The second difference is that instead of returning false when the while loop exits in the iterative version, in case of the recursive version, the condition of low > high is checked at the beginning of the next level of recursion and acts as the boundary condition for stopping further recursive calls by returning false.
* Also, note that the recursive invocations of binarySearch() return back the search result up the recursive call stack so that true or false return value is passed back up the call stack without any further processing.

**ALGORITHM:**

Algorithm BinSrch ( a , I , l , x )

//Given an array a[i:l] of elements in non-decreasing order,

//1 ≤ i ≤ l, determine whether x is present, and

//if so, return j such that x = a[j]; else return 0.

{

if ( l = I ) then // if small(P)

{

if ( x = a[i] ) then return i;

else return 0;

}

else

{ //reduce P into a smaller subproblems.

mid :=[ (i + l ) /2];

if (x = a[mid] ) then return mid;

else if (x < a[mid] ) then

return BinSrch (a , i , mid-1 , x);

else return BinSrch( a, mid+1 , l , x);

}

}

**Time Complexity**

* The time complexity of the binary search algorithm is O(log n).
* The best-case time complexity would be O(1) when the central index would directly match the desired value.
* The worst-case scenario could be the values at either extremity of the list or values not in the list.
* The time complexity of Binary Search can be written as

**T(n) = T(n/2) + c**

* The above recurrence can be solved either using Recurrence Tree method or Master method. It falls in case II of Master Method and solution of the recurrence is :
  + According to master’s theorem

**a=1,b=2**

**n^(log(base b)a)=n^(0) =1**

**T(n) = n^(0)\*U(n)**

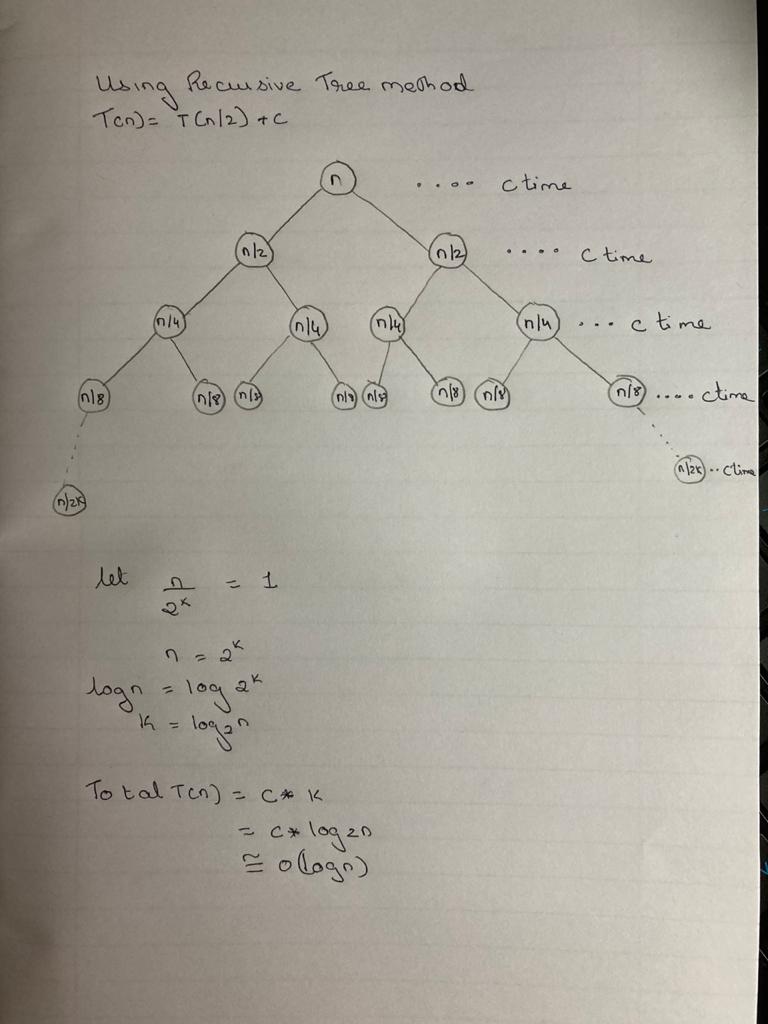
**U(n)->h(n)=F(n)/(n^0)**

**= c/(n^0) =>c=>r=0**

**=>i=0**

**(Log(n))^i+1/(i+1) =(Log(n))**

* + Therefore, the time complexity is T(n) = 1\*Log(n)=O(Log(n))



**Problem Tracing:**

* Let us take an input array a of size =5 and search for element 2 in the array

a=[1,2,3,4,5] n=5 x=2

* For Iterative Binary search:

int found=BinSearch(a,n,x); ***stepcount=1 //for assignment***

//Inside BinSearch() function

int low=0,high=n-1; ***stepcount=3 //for assignment of low and high***

iteration 1: 0<5 true

***stepcount=4 //for while loop condition***

mid=(low+high)/2=(0+5)/2=2 ***stepcount=5 //for assignment of mid***

***stepcount=6 //for if condition***

a[mid]=a[2]=3 and 2<3 therefore, else if is true

***stepcount=7 //for else if condition***

high=mid-1 => high=2-1=1 ***stepcount=8 //for assignment of high***

iteration 2: 0<1 true

***stepcount=9 //for while loop condition***

mid=(low+high)/2=(0+1)/2=0 ***stepcount=10 //for assignment of mid***

***stepcount=11 //for if condition***

a[mid]=a[0]=1 and 2>1 therefore, else will get executed

low=mid+1 =>low= 0+1=1 ***stepcount=12 //for assignment of low***

iteration 3: 1==1

***stepcount=13 //for while loop condition***

mid=(low+high)/2=(1+1)/2=1 ***stepcount=14 //for assignment of mid***

***stepcount=15 //for if condition***

a[mid]=a[1]=2 and 2==2 found!

Return 1;  ***Stepcount=16 //for return statement***

//inside main() function

Found=2

***Stepcount=17 //for if condition***

found ! = 0 therefore else condition will get executed

***stepcount=18 //for executing cout statement***

2 is found at position 1 in array.

**Stepcount=17**

For Recursive Binary Search:

int found=BinSrch(a,0,n-1,x); stepcount=1 //for assignment

//inside BinSrch() function

i=0 l=n-1=4 x=2

***stepcount=2 //for if condition***

i!=l therefore else part will get executed

mid=(i+l)/2 => mid= (0+4)/2=2 stepcount=3 //for assignment of mid

***stepcount=4 //for if condition***

a[mid]=a[2]=3 and 2<3 therefore else if condition true

***stepcount=5 //for return***

return BinSrch(a,0,1,x);

//inside BinSrch(a,0,1,x) function call

i=0 l=1 x=2

***stepcount=6 //for if condition***

i!=l therefore else statements will get executed

mid=(i+l)/2 => mid= (0+1)/2=0 stepcount=7 //for assignment of mid

***stepcount=8 //for if condition***

a[mid]=a[0]=1 and 2>1 therefore else statement will get executed

***stepcount=9 //for return***

return BinSrch(a,1,1,x);

//inside BinSrch(a,1,1,x) function call

i=1 l=1 x=2

***stepcount=10 //for if condition***

i==l therefore if statements will get executed

***stepcount=11 //for if condition***

a[i]=a[1]=2 and 2==2 found!

***stepcount=12 //for return***

return 1;

//inside main() function

found=1

***stepcount=13 //for if statement***

found !=0

***stepcount=14 //for executing cout statement***

* + - **2 is found at position 1 in the array**

**PROGRAM:**

#include<iostream>

using namespace std;

int count = 0;

int BinSrch(int arr[],int first,int last,int element)

{

count++;

if(last==first)

{

count++;

if (element==arr[first])

{

count++;

return first;

}

else

{

count++;

return 0;

}

}

else

{

count++;

int mid=(first+last)/2;

count++;

if(element==arr[mid])

{

count++;

return mid;

}

else if (element<arr[mid])

{

count++;

return BinSrch(arr,first,mid-1,element);

}

else

{

count++;

return BinSrch(arr,mid+1,last,element);

}

}

count++;

return -1;

}

int binsrchi(int arr[],int last, int x)

{

count++;

int low=1;

count++;

int high =last;

count++;

while(low<=high)

{

count++;

int mid = (low+high)/2;

count++;

if(x<arr[mid])

return high = mid - 1;

else if(x>arr[mid])

return low =mid +1;

else return mid;

}

}

int main()

{

count++;

int a[]={-30,-40,50,60,70,80,90};

count++;

int n = 7;

count++;

int x = 70;

count++;

int y = 50;

count++;

int element\_found = BinSrch(a,0,n-1,x)+1;

count++;

if(element\_found == -1 )

{

count++;

cout<<"Element Not Found";

}

else

{

cout<<"-----------------"<<endl;

cout<<"Recursion Methord"<<endl;

cout<<"-----------------"<<endl;

cout<<endl<<"Element Found at Position = "<<element\_found<<endl;

cout<<endl<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Recursive Count = "<<count<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

count++;

int output2 = binsrchi(a,n,y);

cout<<"-----------------"<<endl;

cout<<"Iterative Methord"<<endl;

cout<<"-----------------"<<endl;

cout<<endl<<"Element Found at Position = "<<output2<<endl;

cout<<endl<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

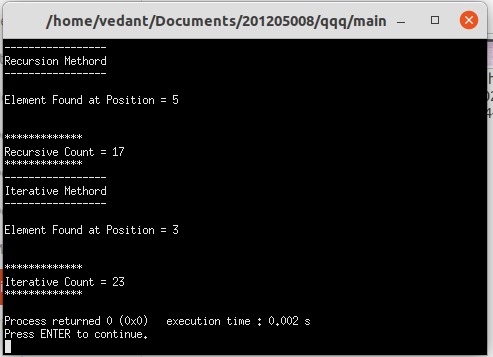
cout<<"Iterative Count = "<<count<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

return 0;

}

**OUTPUT:**



**Conclusion:**

* Detailed concept of Iterative and Recursive Binary Search using Divide and Conquer methods was studied successfully.
* Iterative and Recursive Binary Search Programs were executed successfully implemented.