Binary search

Very efficient algorithm for searching of an element in sorted array:
If *Key* = A[*m*], stop (successful search); otherwise, continue searching by the same method in A[0..*m*-1] if *Key* < A[*m*] and in A[*m*+1 *n* 1] if *Key* > 1..*n*-A[*m*]

```
Binary_search(arr , start_index,  last_index, key)
{     if(start_index==last_index)
          {
          if(key==arr[start_index])         -----------O(1)
                    {
                    return(start_index);
                    }
         else       {
                    retrun(-1)   // not found
                    }
          }
   else
           {        mid=(start_index+ last_index)/2;
                    if(arr[mid]==key)                    -------------------------- O(1)
                    {
                    return(mid);
                    }
                    else if (arr[mid]>key)---------------------O(1)
                    {
                    Binary_search(arr,start_index, mid-1, key);--------------- T(n/2)
                    }
                    else if (arr[mid]<key)                                    OR
                    {
                    Binary_search(arr,start_index, mid+1, key);-----------------T(n/2)
                    }
          }
}
```

Recurrence relation equation

$$T(n)=\begin{cases} O(1) & \text{if } n=1 \\ \underbrace{O(1) + O(1) + O(1)}_{\text{C- constant}} + T(n/2) & \text{if } n>1 \end{cases}$$

$T(n)=T(n/2) + C$

Solving recurrence equation by substitution method

$T(n)=T(n/2)+C$

$=(T(n/4)+C)+C$

$=(T(n/8)+C)+2C=T(n/8)+kC$

|

$=T(n/2^k)+kC$ ---------------eq-1 we know $T(1)=O(1)$ from recurrence equation

Solving for k value

$n/2^k =1$

$n=2^k$ -----------------apply both side $\log_2$

$\log_2 n=\log_2 2^k$

$\log_2 n=k \log_2 2$---------------------------- $\log_2 2=1$

$k= \log_2 n$

Now substitute k value in eq-1

$T(n)=T(1)+ \log_2 n*C$

$=O(1)+ \log_2 n*O(1)$

$=O(\log_2 n)+O(1)$

$= O(\log_2 n)$

Time complexity of **binary search**= $O(\log_2 n)$