DAA-LAB ASSIGNMENT - 2

ALGORITHM:

1) LINEAR SEARCH:

~~// Input: Enter Array~~

```
LSearch (arr[], n, key)
// Input: Enter array to be searched (arr[])
          Enter length of array (n)
          Enter key to be searched in array
          (key)
// Output: Index of key in arr[] if present
           else -1

for (i=0; i<n; i++)
    if (arr[i] == key)
        return i
return -1
```

2) BINARY SEARCH:

```
BSearch (arr[], start, end, key)
// Input: Enter sorted array to be searched (arr[])
          Enter index of 1st element of array (start)
          Enter index of last element of array (end)
          Enter key to be searched in array (key)
// Output: Index of key in arr[] if present
           else -1
```

```
if (start ≤ end)
    mid = (start + end) / 2

    if (arr[mid] == key)
        return mid
    else if (arr[mid] > key)
        return BSearch (arr[], start, mid-1, key)
    else
        return BSearch (arr[], mid+1, end, key)
return -1
```

ok

PAGE No.	
DATE	/ /

TESTCASES: (LINEAR SEARCH)

(+ve) 1. arr[] = {6, 7, 8, 9, 10}

key = 8

n = 5

Output: Index at which 8 is located is 2

(+ve) 2. arr[] = {9, 2, 1, 100, 0}

key = 2

n = 5

Output: Index at which 2 is located is 1

(-ve) 3. arr[] = {1, 3, 4, 6}

key = ~~6~~ 7

n = 4

Output: ~~key~~ 7 not found in array

(-ve) 4. ~~arr[] = {1, 3, 4, 6}~~

~~key = 3~~

~~n = 6~~

~~Output: Size of array is not given properly~~

(+ve) ~~5~~ 4. arr[] = {1, 2}

key = 2

n = 2

Output: Index at which 2 is located is 1

(-ve) 5. arr[] = ~~{1, 3, 4, 6}~~ {}

key = 5

n = ~~4~~ 0

Output: Array is empty.

PAGE No.

DATE / /

TESTCASES: (BINARY SEARCH)

(+ve) 1. arr[] = {1, 2, 3, 4, 5}

start = 0

end = ~~5~~ 4

key = 4

Output: Index at which 4 is found is ~~2~~ 3

(+ve) 2. arr[] = {50, 200, 350, 900}

start = 0

end = 3

key = 50

Output: Index at which 50 is found is 0

(-ve) 3. arr[] = {3, 6, 4}

start = 0

end = 2

key = 6

Output: Array not sorted

α

~~4. arr[] = {10, 20, 21, 40, 51}~~

~~start = 5~~

~~end = 2~~

~~~~Output: 3~~ key = 21~~

~~Output: ~~Initial~~ Values of start & end are wrong.~~

(-ve) 4. arr[] = {10, 20, 21, 51}

start = 0

end = ~~4~~ 3

~~Output~~: key = 30

Output: 30 not found in array.
~~~~

(-ve) 5. arr[] = {1, 3, 7}

start = 0

end = 2

key = 9

output : ~~Element~~ 9 not found in array

TIME COMPLEXITY:

1. LINEAR SEARCH:

- Input's size is the number of elements in array, i.e, n.
- Denote ~~C(n)~~ $C_{worst}(n)$, the number of times this comparision is executed in the worst case
- ~~Algorithm makes one compar~~
- Algorithm has 2 worst-case inputs,
  - * Input in which element to be found is in last index
  - * Input in which element to be found is not in array

$$C_{worst}(n) = \sum_{0}^{n-1} 1 = n-1+1 = n \in O(n)$$

∴, Time complexity of Linear search is $O(n)$

2. BINARY SEARCH:

- Input's size is the number of elements in array, i.e, $n = end - start + 1$
- Algorithm's basic operation = Addition & Division
- Let $A(n)$ be no. of ~~additions~~ calculations done by algorithm.

- Recurrence relation of $A(n)$ will be:

$$A(n) = A\left(\frac{n}{2}\right) + 1$$

as ~~A(n)~~ we check 1 value of mid each call & then cut the array in half either to the left or right of mid & search again for an array of half size, i.e. $A\left(\frac{n}{2}\right)$

- we know that for an array of size 1 the time will be 1 as we only have to check 1 value

so, $A(1) = 1$

- Consider $n = 2^k$

$A(2^k) = A(2^k/2) + 1$

$A(2^k) = A(2^{k-1}) + 1$

& $A(1) = 1$

By Backward substitutions

$A(2^k) = A(2^{k-1}) + 1$

Put $A(2^{k-1}) = A(2^{k-2}) + 1$

$A(2^k) = A(2^{k-2}) + 2$

Put $A(2^{k-2}) = A(2^{k-3}) + 1$

$A(2^k) = A(2^{k-3}) + 3$

$\therefore$, By applying these we get

$A(2^k) = A(2^{k-k}) + k$

$A(2^k) = A(1) + k$

$A(2^k) = k + 1$

Resubbing $2^k = n$ & $k = \log_2 n$

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

- $\therefore$, Time complexity of Binary search is $O(\log_2 n)$