Mani S Palinkar
23107036 Batch B

Linear Search

NAME: LinearSearch(arr, key)
// Input: Array 'arr' of numbers, number whose index has to be found (key)
// Output: Index of number, if found

```
for(i from 0 to n-1)          ---- 1xn
  {
    if arr[i] == key          ---- 1xn
      return i;               ---> 1
  }
return -1;                    ---> 1
```

Test cases

- **Positive**

1) Array :- 1 2 3 4 5
   key : 2
   O/P : 1

2) Array :- 3 6 10 -5 -20
   key : -5
   O/P : 3

3) Array :- 5 10 50 1000 2000
   key : 2000
   O/P : 4

4) Array : -1 -2 -3 100 200
key : -3
O/P :- 2

- Negative

1) Array :- 1 2 3 4 5
key :- 6
O/P :- -1

2) Array :- -5 -15 -50 -60
key :- -10
O/P :- -1

3) Array :- -100 -200 -300 -400
key :- -500
O/P :- -1

4) Array :- 200 300 500 600 700
key :- 400
O/P :- -1

Time complexity

Time eqn = 2n+2.

∴ Worst case time complexity :- O(n)

Saathi

Date __/__/____

Binary Search (Recursive)

Name :- BinarySearch (arr, key, low, high)
// Input :- Sorted array, key
// Output :- Index of key, if found.

```
low = arr[0];
High = arr[1];
BinarySearch ( arr, key, low, high) {
  if (high >= low) {
      mid = (low + high)/2 }

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

else if (arr[mid] > key) {
      BinarySearch (arr, key, low,
                    mid - 1);

}

else if
   BinarySearch ( arr, key, mid+1,
                  key);

}

else
return -1;
}
```

Test cases

- **Positive**

1) Array :- 3 6 9 12 15
   key :- 6
   O/P :- 1

2) Array :- 1 2 3 4 5
   key :- 1
   O/P :- 0

3) Array :- 6 12 18 27 30
   key :- 18
   O/P :- 2

4) Array :- -10 -20 2 10 -100
   key :- -100
   O/P :- 4

- **Negative**

1) Array :- 1 3 5 7 9
   key :- 11
   O/P :- -1

2) Array :- 3 6 10 9 11
   key :- 1000
   O/P :- -1

3) Array :- 5 10 100 25 250
Key :- 1000
O/P :- -1

4) Array :- 10 20 30 40 50
Key :- 500
O/P :- -1

Time complexity

Binary Search Recursive

In every step, we reduce the array in which we have to search by $n/2$.

In other words, at every step k, we need to search through an array of size at most $n/2^k$. And we need to find smallest k for which we have no subarray to search through

$$\frac{n}{2^k} < 1$$

$$k = \log_2 n + 1$$

Worst case :- $O(\log n)$

Date __/__/____

Saathi

Conclusion :- We conducted 2 experiments, linear as well as recursive binary search. We wrote pseudo code and test cases for both of them and calculated their worst case time complexity.

Page No.