Searching Algorithms

· Linear Search – Unsorted Input
· Linear Search – Sorted Input
· Binary Search (Sorted Input)
· String Search: Tries, Suffix Trees, Ternary Search.
· Hashing and Symbol Tables

Linear Search – Unsorted Input

```
int linearSearchUnsorted(int arr[], int size, int value)
{
        int i = 0;
         for(i = 0 ; i < size ; i++){
                 if( value == arr[i]) return i;
        }
         return -1;
}
```

Time Complexity: O(n). As we need to traverse the complete array in worst case. Worst case is when your desired element is at the last position of the array. Here, 'n' is the size of the array.
Space Complexity: O(1). No extra memory is used to allocate the array.

Linear Search – Sorted

```
int linearSearchSorted(int arr[], int size, int value)
{
        int i = 0;
         for(i = 0 ; i < size ; i++){
                 if( value == arr[i]) return i;
                else if(value < arr[i]) return -1;
        }
         return -1;
```


Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Linear Search – Sorted (cont)

```
>}
```

Time Complexity: O(n). As we need to traverse the complete array in worst case. Worst case is when your desired element is at the last position of the sorted array. However, in the average case this
algorithm is more efficient even though the growth rate is same as unsorted.
Space Complexity: O(1). No extra memory is used to allocate the array.

Binary Search

```
/ Binary Search
Algorithm – Iterative
Way /
int Binary sea rch(int
arr[], int size, int
value)
{
       int low = 0;
       int high = size-
1;
       int mid;
        whi le(low <=
high){
               mid = low
+ (high- low)/2; / To
avoid the overflow /
               if
(arr[mid] == value)
return mid;
               else if
(arr[mid] < value) low =
mid + 1;
               else high
= mid - 1;
       }
        return -1;
}
```

Time Complexity: O(logn). We always take half input and throwing out the other half. So the recurrencerelation for binary search is T(n) = T(n/2) + c. Using master theorem (divide and conquer), we get T(n) =O(logn)
Space Complexity: O(1)

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com