02/12/21

→ LINEAR SEARCH

*Searching: finding given value/position in a list of values.

*linear/sequential search: Basic & simple, compares the target value with elements of list to identify the given value to find.

Time Complexity

* linear search me one by one value pe jaake check kiya jaata hai w.r.t underlying condition.

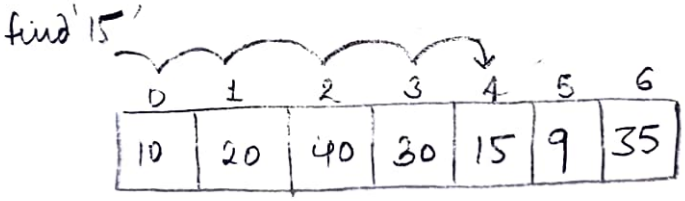
*Best case – No. of checks will the loop make ~~in~~ best case, the element ~~[illegible]~~ found at 0th index ~~[illegible]~~ so one comparison is done that is best case.
ek array me jitne kam check karne pade hai ek position pe wo best case hai

*worst case – loop will go through every element & then it can ~~be~~ say element not found.

code = condition (if-else)
loop/iteration

*practice questions from github

*Algorithm:-
1) Traversal of array.
2) Compare all elements one by one with targeted key.
3) If element matches key, return current index.



0	1	2	3	4	5	6
10	20	40	30	15	9	35

→ BINARY SEARCH

* sortedarrays < ascending / decendending } orders

* search in sorted array which repeatedly divide the search interval in half.

* idea of binary search to reduce time complexity.

*

* **ALGORITHM:**

i) find the middle element < $\frac{\text{start index} + \text{end index}}{2}$ / for large range index — $\text{start} + \frac{(\text{end} - \text{start})}{2}$

ii) Take the middle element

kept repeating until search is done

ascending order array { , target element > middle → search right part
, target element < middle → search left
, target == middle → answer

* if start index > end index, element not found

⇒ To check whether array is sorted in ascending or (descending.

ORDER AGNOSTIC BINARY SEARCH } compare start index value to end index value.

if $s > f$ → increasing / ascending
else → decreasing / descending

iii) for descending sorted array,

target element > middle → search left part
target element < middle → search right part

* **Time complexity:** (abhi samajh nhi ayega)

Best case : o(1)

worst case : o(logn)

↳ : Total no. of comparisons in worst case ⇒ $\boxed{k = \log_2 N}$ } size of array

Pseudocode for Binary Search

```
//whether array is sorted in ascending or descending
boolean checkArrayOrder = arr[start] < arr[end];
```

if this condition is true, so ascending and if false then descending

RECURSIVE IMPLEMENTATION

```
//function
int binarySearch(int arr, int start,
                 int end, int key){
 int mid = (start+end)/2; //calculate mid
 or
 mid = (start+(end-start))/2;

 [illegible]
 if (mid == key) {
    return mid;
 }
 if (checkArrayOrder){
   if (mid >key){                              ascending
     return binarySearch(arr, start,
                          mid-1, key);
   }
   else{
     return binarySearch(arr, mid+1,
                          end, key);
   }
 }
 else{
   if (key > mid){                             descending
     return binarySearch(arr, start,
                          mid-1, key);
   }
   else{
     return binarySearch(arr, mid+1,
                          end, key);
 }
```

Iterative Implementation

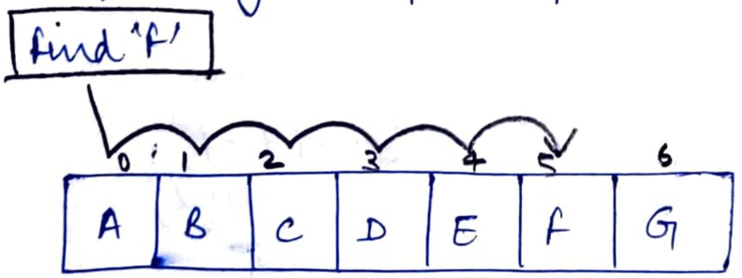
```
while (stard <=end){
 calculate mid term
 if (mid == key) {
    return mid;}
 if (checkArray Order){
   if (key <mid){
     end = mid-1;                    ascend.
     //start=same;
   }
   else{
     //end =same;
     start = mid +1;
   }
 }
 else{
   if (key>mid){
     end = mid -1;                   descend
     //start = same;
   }
   else {
     start = mid+1;
     //end = same;
   }
}}
```

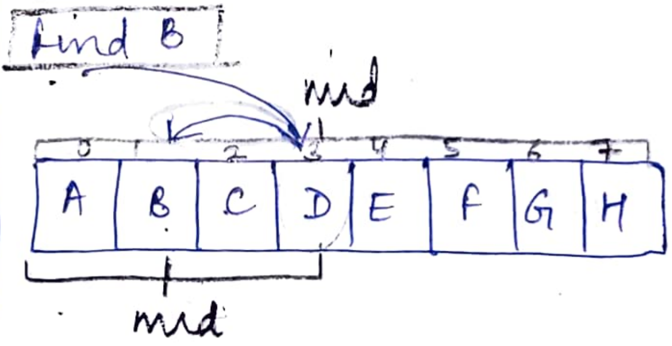
* when to use Binary search

- → sorted array
- → when the problem stated is in sequential particular pattern/way so that it follows a sequential pattern to get answer.

⇒ Difference b/w Linear & Binary Search

LINEAR SEARCH	BINARY SEARCH
* Input data need not to be sorted.	* Input data needs to be sorted.
* It does sequential access	* It access data randomly
* Time complexity → $O(n)$	* Time complexity → $O(\log n)$
* equality no. of comparisons	* performs order no. of comparisons.


find 'F'
0
1
2
3
4
5
6
A
B
C
D
E
F
G


find B
mid
0
1
2
3
4
5
6
7
A
B
C
D
E
F
G
H
mid