LINEAR SEARCH ALGORITHM

- What is searching ?

arr = [18, 12, 9, 14, 77, 50] (unsorted) (size = 6)

Q. Find whether (14) exists in array or not.

If no value found, return -1.

How many checks will the loop make in best case i.e element found at 0th index!

arr = [8, 9, 12, ---- . 200 elements]

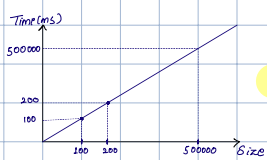
Target = 8

→ 1 comparison in best case

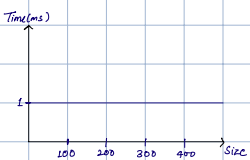
Worst Case: → You do not find the target item

iterate / go through every item and then in the end it says did not find it.

if size of array = 100 ⟹ 100 comp^n [100 ms]
= 200 ⟹ 200 " [200 ms]
= 1 lakh ⟹ 1 lakh " [100000 ms]



{Linear Time Complexity}



{Constant Time Complexity}

Worst Case

500 items ≃ 500 ms
(by analysing the graph)

Best Case

item found at 0th index then time does not depends on size of array

it will run & take time till 0th index only whatever the size of array be.

Q. Search for (3) in the range of index [1, 4]

arr = [18, 12, -7, 3, 14, 28]

Q. Search in 2D-Array

$$arr = \begin{bmatrix} [1, 2, 3] \\ [9, 18, 5] \\ [6, 7, 14] \end{bmatrix}$$

```
for (row=0; row<length(array); row++) {
    for (col=0; col<length(row); col++) {
        if (arr[row][col] = Target) {
            // found ans
        }
    }
}
```

Q. Find no. of nos. that have even no. of digits.

nums = [18, 124, 9, 1764, 98, 1]

① Count the no. of digits

② Convert the no. to String & take the length

eg:- 1764 ⟹ "1764" ⟶ take the length

```
while (n>0) {
    count++
    n = n/10
}
```

BINARY SEARCH ALGORITHM

→ usually used for sorted arrays

arr = [2, 4, 6, 9, 11, 12, 14, 20, 36, 48] (indices 0–9)

target = 36

1) Find the middle element

2) target > middle → search in the right

else search in left

3) if middle element == target element ⟹ Ans

arr = [2, 4, 6, 9, (11), | 12, 14, (20), (36), 48] → [12, 14, (20), 36, 48] → [36, 48]

(s = 0, m = 4, e = 9; s' = 5, m' = 7; s'' = 8, m'' = 8, e'' = 9)

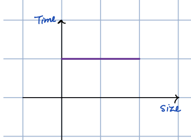
↓ middle element

$mid = \frac{s+e}{2}$, $mid = \frac{0+9}{2} = 4 \equiv 11$ → 36 > 11 → search in right of middle element

$mid' = \frac{5+9}{2} = 7 \equiv 20$ → 36 > 20 → search in right of middle element

$mid'' = \frac{8+9}{2} = 8 \equiv 36$ → target ⟹ Ans → index = 8

→ if $s > e$:→ element not found

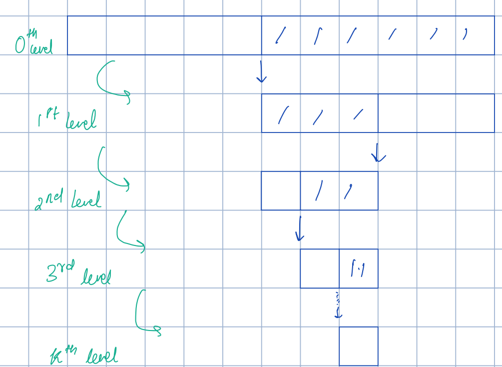


Best Case O(1)

element found at first step
(ie, first middle element = Ans)

• Why Binary Search?

Q) Find the max number of such comparisons in worst case.



0th level	N	$\frac{N}{2^0}$
1st level	$\frac{N}{2}$	$\frac{N}{2^1}$
2nd level	$\frac{N/2}{2} = \frac{N}{4}$	$\frac{N}{2^2}$
3rd level	$\frac{N/4}{2} = \frac{N}{8}$	$\frac{N}{2^3}$
kth level	$1 =$	$\frac{N}{2^k}$

$\therefore \frac{N}{2^k} = 1$, k → total no. of levels/comparisons in the worst case

$\Rightarrow 2^k = N \Rightarrow k \ln 2 = \ln N \Rightarrow \boxed{k = \log_2 N}$

→ Total comparisons in the worst case = $\log_2 N$

Search in a array of 1million → 1000000

Linear Search	Binary Search
In worst case → 1million comparisons	In worst case, No of comparisons = $\log_2(10^6)$ ≈ 20 comparisons only

- Best way to find middle element :→

mid = $\frac{start + end}{2}$ ⟹ This may exceeds the range of integers in java

mid = start + $\frac{(end - start)}{2}$ → $s + \frac{e-s}{2} = \frac{2s + e - s}{2} = \frac{s+e}{2}$

- Order-Agnostic Binary Search :→ { when we don't know the Order of an Array whether it is Asc. or Desc. }

arr = [90 (s, 0), 75 (1), 18 (2), 12 (m, 3), 6 (4), 4 (5), 3 (e, 6), 1 (7)] → Sorted in Descending Order

target = 75

→ if target > middle → search in left of middle element

e = m-1

→ if target < middle → search in right of middle element

s = m+1

→ if e > s ⟹ Increasing Order Sorted (e − s = no. of elements)

else ⟹ Decreasing Order Sorted

* Ceiling :→ Smallest element in an array greater or equal to target.

eg:→ arr = [2, 3, 5, 9, 14, 16, 18] { Since, it is Sorted Array ⟹ use Binary Search }

Ceiling(arr, target = 14) = 14

Ceiling(arr, target = 15) = 16

arr = [2 (s, 0), 3 (1), 5 (2), 9 (m, 3), 14 (4), 16 (e, 5), 18 (6)] target = 15 → Ceiling = 16

1 5 7 9 → s = m+1 → [14 (s, 4), 16 (m, 5), 18 (e, 6)]

16 < 16 → e = m-1 → [14 (s = m = e, 4)]

15 > 14 → s = m+1 → [14 (e, 4), 16 (s, 5), 18 (6)] { → target > mid } ⟹ Ans ⟹ element at index of start → 16

Condition for breaking of loop → while (start ≤ end)

when while loop (s ≤ e) breaks ⟹ [start = end + 1]

→ Use of start & end pointer :→ it means Ans will be b/w 's' & 'e' → s ≤ Ans ≤ e

* Floor :→ biggest number in an array smaller or equal to the target

eg:- arr = [2 (s, 0), 3 (1), 5 (2), 9 (m, 3), 14 (4), 16 (5), 18 (e, 6)]

floor(arr, target = 15) = 14

floor(arr, target = 14) = 14

when loop breaks (s > e) → same thing as before, just return end → instead of -1, if no not found

[s target e] not b/w s & e

[e (Ans) s] { s = e+1 } (s > e)

Q] Find the peak index in a Mountained (Bitonic) Array

→ Mountained (Bitonic) Array → decreases first then increases / increases first then decreases

eg:- 1 2 3 5 (7) 6 3 2

↑ses → ← ↓ses

find this number

eg:- [1, 2, 3, 5, 6, 4, 3, 2] (indices 0–7; s = 0, m = 3, e = 7)

• Case I:→ if element[mid] > element[mid+1] ⟹ You are in ↓sing part of array

→ e = mid —because→ here we don't know element[mid-1] > element[mid]

// check in left side

• Case II:→ if element[mid] < element[mid+1] ⟹ You are in ↑sing part of array

→ s = mid + 1 —because 2→ here we know that element[mid] < element[mid+1]

•→ when will loop break? ⟹ when s = e

arr = [1, 2, 3, 5, 6, 4, 3, 2] (indices 0–7; s = 0, m = 3, e = 7)

arr[m] < arr[m+1] → Case II: → s = m+1

[6, 4, 3, 2] (indices 4–7; s = 4, m = 5, e = 7)

Case I: ↓

[6, 4] (indices 4–5; m = s = 4, e = 5)

Case I: ↓

[6] (m, s, e) ⟹ In the end, "s & e" both points to the largest no. ⟹ Ans.

Q] Find in Mountain Array

arr = [1, 2, 3, 4, 5, 3, 1]

target = 3

→ arr = [1, 2, 3, 4, 5, 3, 1] (indices 0–6)

i) Find peak element ⟹ 4th Index

ii) Binary Search in Ascd. Array ⟹ (0, 4)

iii) If not found, Binary Search in Descd. Array ⟹ (4, 6)

Q] Search in Rotated Sorted Array

arr = [2, 4, 5, 7, 8, 9, 10, 12]

→ arr = [2, 4, 5, 7, 8, 9, 10, 12] (indices 0–7)

After 1 rotation :-

arr = [12, 2, 4, 5, 7, 8, 9, 10] (indices 0–7)

After 2nd rotation :-

arr = [10, (12), 2, 4, 5, 7, 8, 9] (indices 0–7)

↓

pivot

- Approach 1 :-

i) Find the pivot in the array.

→ Pivot → from where your next numbers are Ascending

[3, 4, 5, 6, (7), 0, 1, 2]

3, 4, 5, 6 : Asc. ; 7 : Pivot ; 0, 1, 2 : Asc.

ii) Search in first half → Simple BS (0, Pivot)

Otherwise, Search in second half → Simple BS (Pivot+1, end)

→ Finding Pivot :-

[3, 4, 5, 6, 7, 0, 1, 2] (s=0, m=3, e=7)

Case I :- [3, 4, 5, 6, 7, 0, 1, 2] (m=4, s=5)

only these 2 will be using

→ when you find that mid > mid+1 element : i.e pivot [Ans]

Case II :- [3, 4, 5, 6, 7, 0, 1, 2] (m=5)

→ if mid element < (mid-1) element i.e → Pivot = (m-1) element [Ans]

Case III :- [... 4, 5, 6, 3, 2, 1, 0 ...] (s=0, e=2, m=3, s=4, e=6)

smaller than s

→ if start element > mid element

In this case, all elements from 'mid' will be smaller than start

Hence, we can ignore all these elements, since we are looking for peak i.e largest element → Pivot

⇒ e = mid - 1

Case IV :- [3, 4, 5, 6, 2] (s=0, m=2, s=3, e=4)

larger than mid element { if 'm' was Pivot → it would have been returned in case II }

→ if start element < mid element

Start = mid + 1

Hence proved that bigger numbers lie ahead

So, ignore mid & put $\boxed{s = m+1}$

Now

→ Finding Target :-

arr = [4, 5, 6, 7, 0, 1, 2] (s=0, p=3, e=6)

Case I :- Pivot element = target ⇒ Ans

Case II :- Target > Start element ⇒ eg:- Search for '6' → Search Space = (s, p-1)

why? → All numbers after Pivot are smaller than start

Case III:→ Target < Start element

i.e; we know that all elements from start to pivot are going to be bigger than target.

eg:- Search for Target = 1 → all element b/w (s, p) → greater than '1' ⇒ Search space = (p+1, e)

↦ For duplicate elements :→

arr = [2, 2, 2, 2, 9]

[2, 9, 2, 2, 2] → 2 Rotations (indices 0 1 2 3 4; s, m, e marked)

we can ignore as these two are same

Q7 Split Array Largest Sum

arr = [7, 2, 5, 10, 8], m = 2

→ arr = [7, 2, 5, 10, 8]

Split		Largest:
[7, 2, 5, 10] (24)	[8] (8)	24
[7, 2, 5] (14)	[10, 8] (18)	(18) ⇒ Ans
[7, 2] (9)	[5, 10, 8] (23)	23
[7] (7)	[2, 5, 10, 8] (25)	25

Case I:→ Min^m no. of partitions that we can make = 1   { m = partitions }

Case II:→ Max^m no. of partitions that we can make = N

arr = [7, 2, 5, 10, 8] ⇒ [7], [2], [5], [10], [8]

Ans for Case I:→ Sum of entire arrays = 32 ⇒ Max value of Ans of Question

Ans for Case II:→ Max^m element of the array = 10 ⇒ Min value of Ans of Question

] ⇒ [10, 32]

min Ans = max value in array

max Ans = Sum of all values in array

[10, 32] → Apply Binary Search here

start = 10   end = 32

mid = $\frac{s+e}{2}$ = $\frac{42}{2}$ = (21)

Try to see, if you can split the array with '21' as the max Sum

	pieces:
[7, 2, 5, 8, 10]	2
[7, 2, 5], [8, 10]	

Check 1:→ if (pieces ≤ m) ⇒ end = mid

[10, 32] → Apply Binary Search here

s = 10, e = 21

mid = 15 → max Sum Should not exceeds mid

[7, 2, 5], [8], [10] → pieces = 3

Check 2:→ if (pieces > m) :→ ⇒ s = mid+1

→ s = mid+1 = 16 , end = 21

mid = $\frac{16+21}{2}$ = 18

[7, 2, 5], [8, 10] → pieces = 2

→ s = 16 , e = 18

mid = $\frac{16+18}{2}$ = 17

[7, 2, 5], [8], [10] → pieces = 3

→ s = mid+1 = 18 , e = 18

mid = $\frac{18+18}{2}$ = 18 ⇒ Ans (s = e)

So, the Ans exists definitely

Hence, by the above two checks → we will reach Ans