

We will start at 7:35

// Searching → It involves, some set of elements & a target that we want to search.

* Linear Search → Consider you have a list of numbers, and you have to search for a target number.

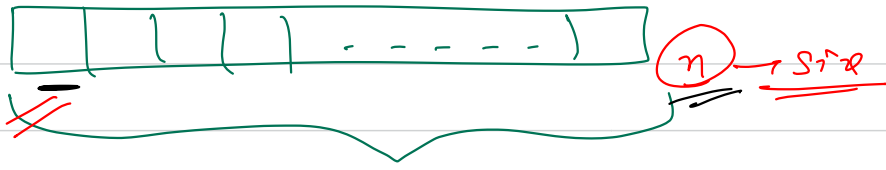
It says iterate one by one on all the elements, & if the current element is equal to the target, return the index of current element.

Search space

target $\rightarrow \underline{\underline{T}}$

region when
you have

the possibility
of finding
target



1 st iteration	\rightarrow	<u>Size of search space</u> n
2 nd iteration	\rightarrow	$n-1$
3 rd iteration	\rightarrow	$n-2$
:		:
:		:
:		:
:		:
<u>n^{th} iteration</u>		<u>1</u>

K steps

$K = n$

2

TC $\rightarrow O(n)$

SC $\rightarrow O(1)$

Worst case scenario \rightarrow when you don't even
have the element present or the element can
be at last index \rightarrow TC \rightarrow $O(n)$

Best Case Scenario \rightarrow element is present at 0th
index \rightarrow $\Omega(1)$

$\{2, 2, 2, 2\}$

3 $O(n)$

a) list of equal elements

b) unsorted list

c) sorted list

d) all of above ✓

→ In linear search we don't care about the permutation of list.

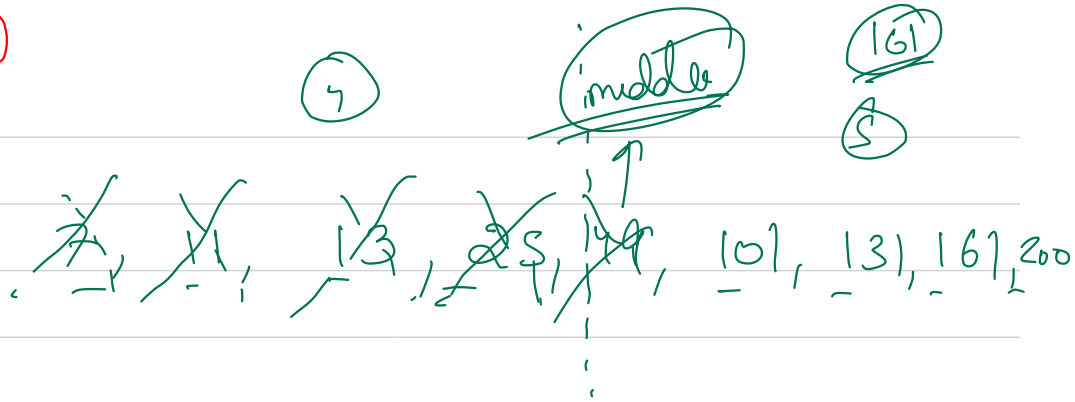
all the words with starting character greater than
(i) will never have centricity

Maybe we can also arrange data in a
specific manner

Binary Search

Some elements which are sorted in increasing order of their values

131 ← target



$n \rightarrow n-1 \rightarrow n-2 \rightarrow \dots$ Linear Search

k terms

$n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8} \rightarrow \dots \rightarrow \frac{n}{2^k}$

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

$$\rightarrow k = \log_2 n$$

$$TC \rightarrow O(\log n)$$

$$SC \rightarrow O(1)$$

$$n \rightarrow 10^3 \longrightarrow \begin{array}{l} O(n) \longrightarrow 10^3 \\ O(\log n) \longrightarrow 10 \end{array}$$

$$n \rightarrow 10^6 \longrightarrow \begin{array}{l} O(n) \approx 10^6 \\ O(\log n) \approx 20 \end{array}$$

$$n \rightarrow 10^9 \longrightarrow \begin{array}{l} O(n) \approx 10^9 \\ O(\log n) \approx \underline{\underline{30}} \end{array}$$


4

5 9 10 13 19 21 27

a) list of equal elements

b) unsorted list

c) sorted list

d) all of above 

Binary Search

Can be applied on all those search conditions when we can divide the search space into 2 parts such that the 2 parts are distinguishable based on some property.

① ← target

→ unsorted list

3, 6, 8, 10, 15, 0, 1, 2

unsorted ← from a sorted list which has
been rotated.

~~X~~ 1 2 3 6 8 10 15 ←

~~X~~ 2, 3, 6, 8, 10, 15, 0 ←

~~X~~ 3, 6, 8, 10, 15, 0, 1

3, 6, 8, 10, 15, 0, 1, 2

```

1 def binary_search(arr, target):
2     # first of all we will define the
3     n = len(arr)
4     left, right = 0, n-1 # boundaries
5     while(left <= right):
6         mid = (left + right) // 2
7         if arr[mid] == target:
8             return mid
9         elif arr[mid] > target:
10            right = mid - 1
11        else:
12            left = mid + 1
13
14    return -1

```

target = 10 ← right

0 1 2 3 4 5 6 7
0, 5, 10, 19, 33, 45, 56, 68
10 ← target

left	right	mid
0	7	3
0	2	1
2	2	2

TC
 $O(1)$
 $O(\log n)$
 $O(\log n)$

SC
 $\rightarrow O(1)$
 $O(1)$
 $O(1)$

$$\boxed{-10^6 - 10^6}$$

$l + r > \text{max}$
 exceed boundary

$$\frac{l + r}{2}$$

$$l, r \rightarrow [\text{min}, \text{max}]$$

$$\begin{aligned}
 l &\rightarrow 2147483640 \\
 r &\rightarrow 2147483642
 \end{aligned}$$



2147483647

→ max

-2147483648

→ min

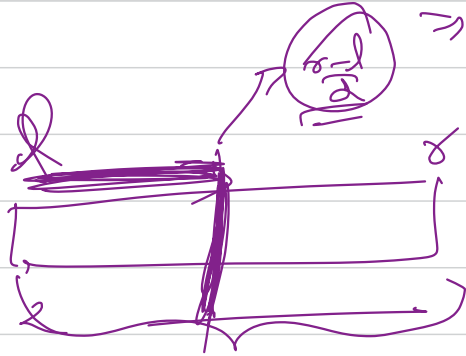


\Rightarrow

$$\underline{\underline{\text{mid}}} \rightarrow l + \frac{r-l}{2}$$
 \rightarrow modified mid

$$\frac{l+r+l-l}{2} \rightarrow \frac{2l+r-l}{2}$$

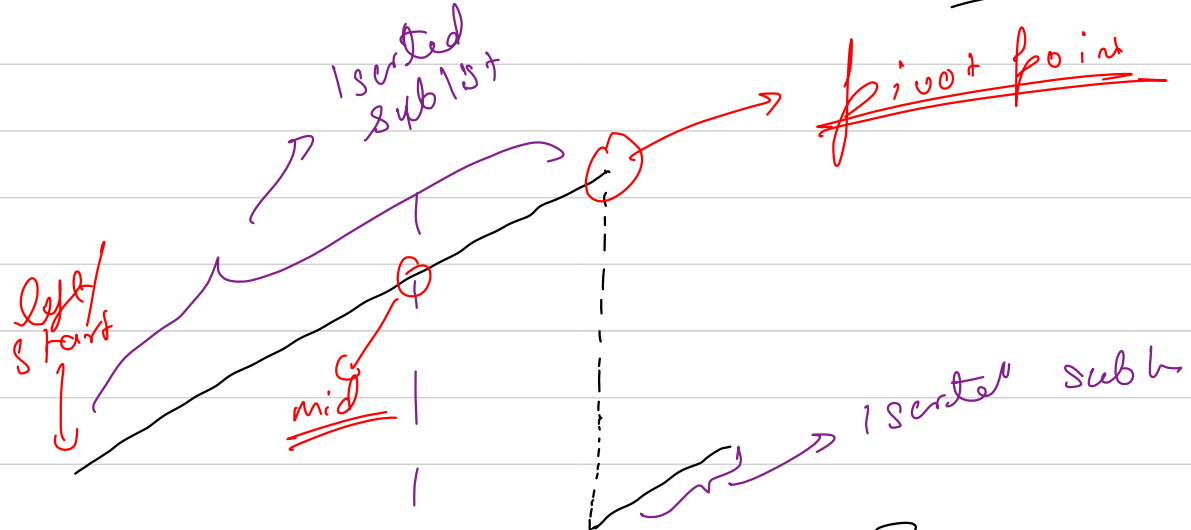
$$l + \frac{r-l}{2}$$



Q.2 Binary Search on sorted rotated list

value n

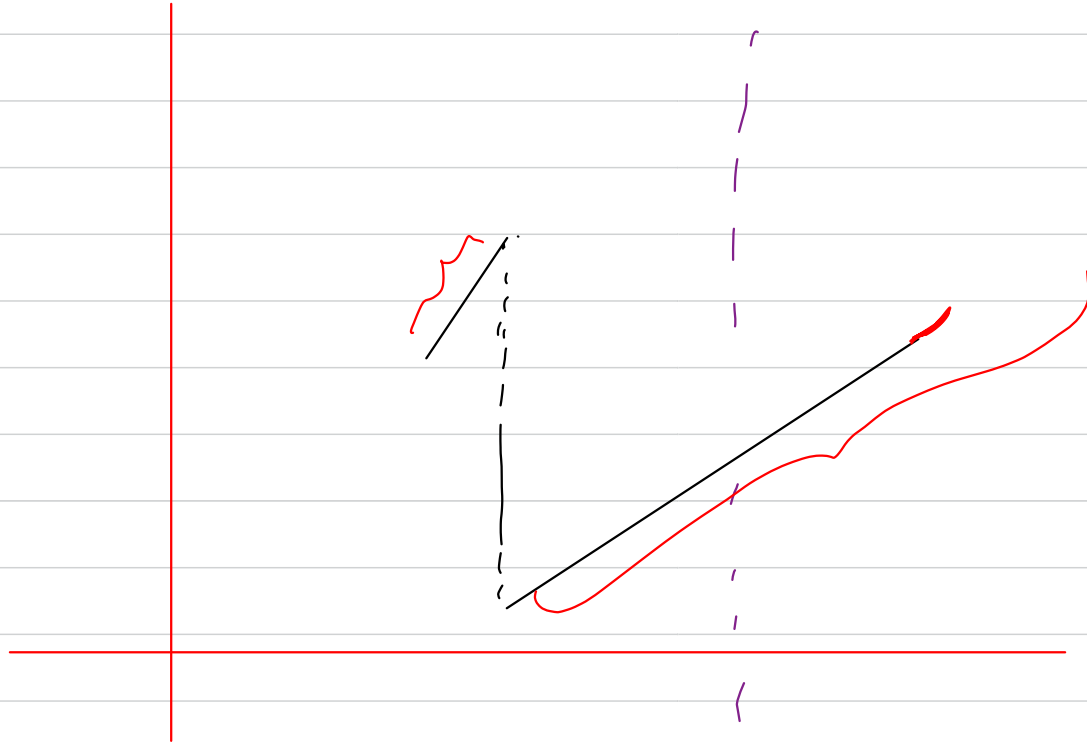
1st of peak



$a[start] < a[mid]$

mid

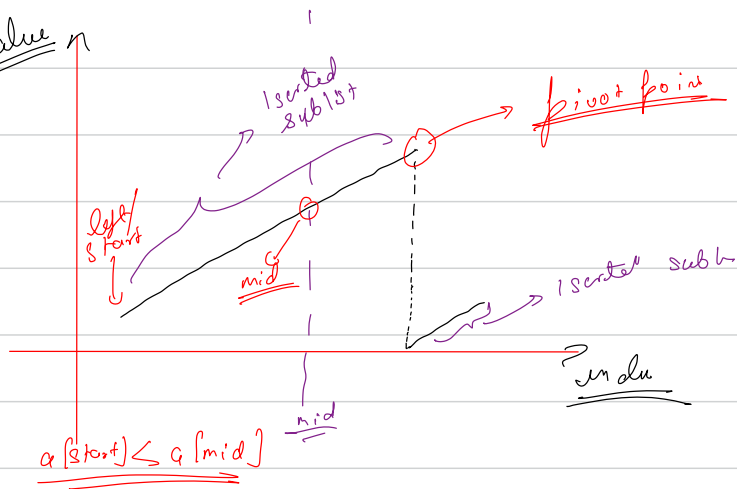
index



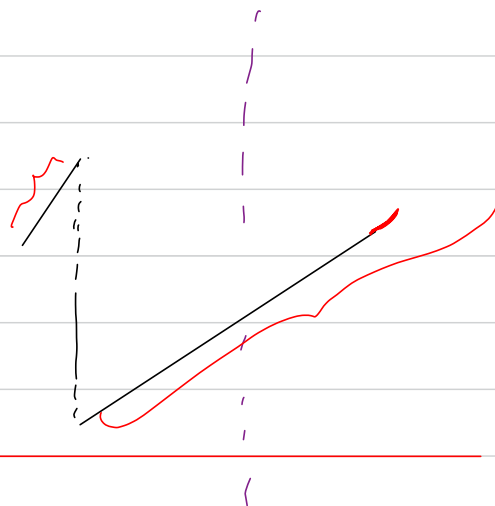
5, 6, 0, 1, 2, 3, 4

$$\underline{a[star] > a[mid]}$$

Case 1
value



Case 2

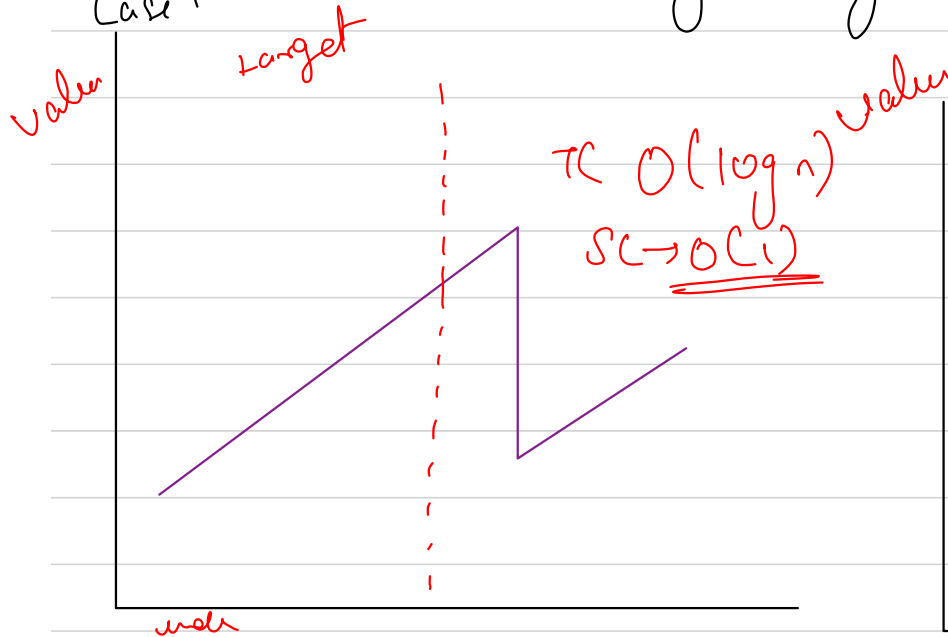


5, 6, 0, 1, 2, 3, 4

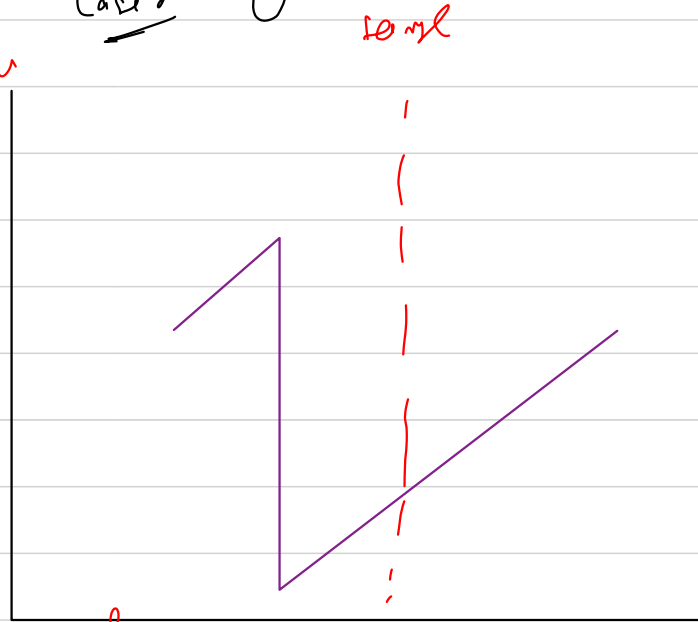
$$\underline{\underline{a[\text{star}] > a[\text{mid}]}}$$

Can we do it using single binary search

Case 1



Case 2



$$arr[start] < arr[mid]$$

$$\rightarrow T \geq arr[s] \text{ and } T \leq arr[mid]$$

$$\hookrightarrow right = mid - 1$$

else

$$\hookrightarrow left = mid + 1$$

and

$$arr[start] > arr[mid]$$

$$T \geq mid \text{ and } T \leq end$$