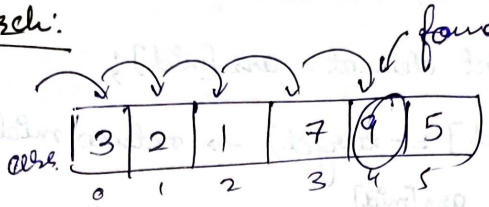
WEEK - 4

SEARCHING AND SORTING

CLASS - I

Linear Search:



target = 9

Code:

```
for (int i=0; i<; i++)
    if (arr[i] == target)
        cout << "found" << endl;
```

Linear complexity

TC = O(n)

SC = O(n)

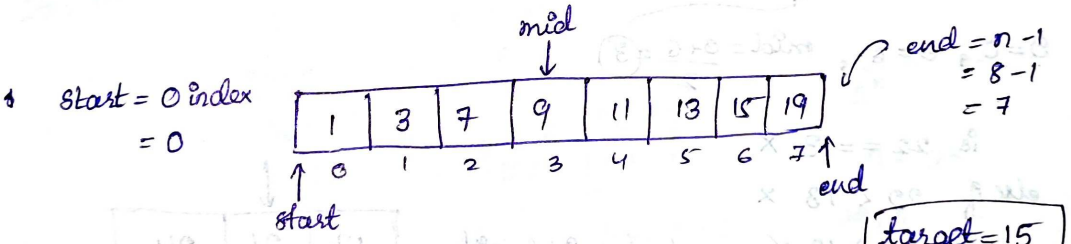
L.S ⇒ 1000 elements = 1000 comparisons.

Binary Search: but In BS ⇒ 1000 elements = 10 comparisons.

but, → **condition:** elements should be monotonic order (~~i.e~~ either increasing / decreasing)

↓

elements should be sorted.



Steps:

① start = 0 index

② end = 7

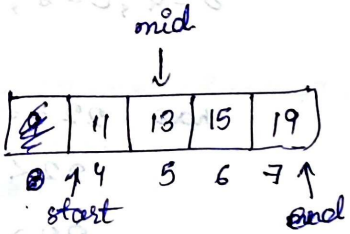
③ mid = $\frac{(start + end)}{2} = \frac{0+7}{2} = 3$

④ mid = 9 < 15 → so search right side.

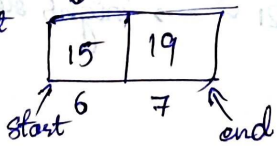
9 = 15 ✗

9 < 15 ✓

9 > 15 ✗



⑤ mid = 13 < 15 → right



start = 6

end = 7

mid = 6

⑥ mid = 15 = target. (else return closest end)

↳ returns index 6

Code of Binary Search: (In pc notes)

```
int start = 0;
int end = size - 1;
int mid = (start + end) / 2;
while (start <= end)
{
    int element = arr[mid];
    if (arr[mid] == target) → return mid;
else if (target < arr[mid]) → end = mid - 1;  ⇒ search left
else → start = mid + 1;  ⇒ search right
    mid = (start + end) / 2;
}
return -1;  → it means 'element not found';
```

Q)

arr

1	3	9	13	17	21	24
0	1	2	3	4	5	6

target = 22

$s = 0, e = 6, mid = \frac{0+6}{2} = 3$

is $22 == 13$ ×

else if $22 < 13$ ×

else $22 > 13$ ✓ → check right side

17	21	24
4	5	6

$s = 4, e = 6, mid = \frac{6+4}{2} = 5$

now $22 == 21$ ×

$22 < 21$ ×

$22 > 21$ ✓ → check right side

24
6

$mid = \left(\frac{start + end}{2}\right)$ ISSUE?

↳ OVERFLOW

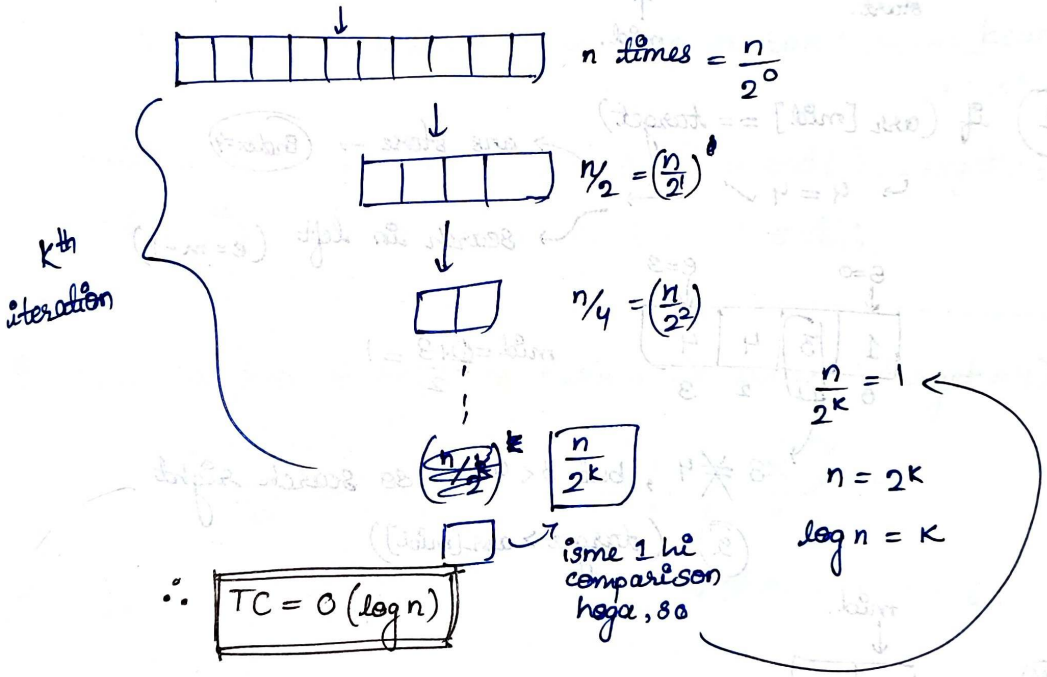
$\Rightarrow$ (-2^{31}) to $(2^{31}-1)$ Range.

if $s = e = 2^{31}-1$ ↳ mid will go out of range.

$\therefore$ use → $\boxed{mid = start + \left(\frac{end - start}{2}\right)}$ → no. chota hi rahega.

$\left(m = s + \frac{(e-s)}{2} = \frac{2s + e - s}{2} \Rightarrow \frac{s+e}{2}\right)$ → same formula.

Time Complexity of Binary Search



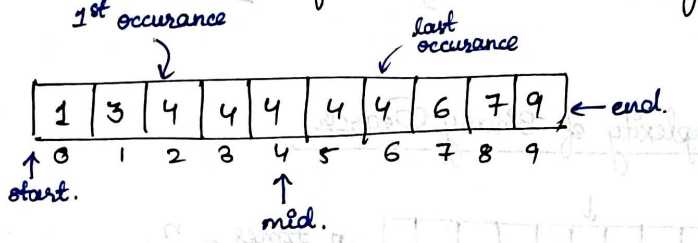
Eg: 1024 size → 512 → 256 → 128 → 64 → 32 → 16 → 8 → 4 → 2 → 1

10 comparisons

Inbuilt function for Binary Search :-

```
vector <int> arr {1,3,5,7,9,11,13,16};
if (binary_search (arr.begin(), arr.end(), 13))
 {
    cout << "Found" << endl;
 }
else {
 }  cout << "Not found" << endl;
```

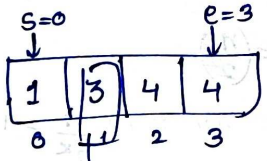
Q) Find the Occurance of an element. (target = 4)



(1) if (arr[mid] == target)

↳ 4 = 4 ✓ → ans store → (index=4)

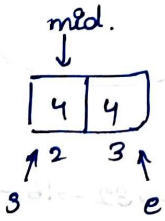
→ search in left (e = m-1)



$mid = \frac{0+3}{2} = 1$

3 ≠ 4, but 3 < 4 → so search right

(2) (target > arr[mid])



4 = 4 ✓ ⇒ ans store → (index = 2)

⇒ search left

(e = m-1 ⇒ 2-1 = 1)

∴ s = 2 & e = 1

s > e → stop here.

(3) Now if (arr[mid] > target)

↳ search left.

Conditions : (in-short summary)

① if (arr[mid] == target)
↳ ans = mid; → store ans
↳ end = mid-1; → To search left

If you are searching Lastoccurance, then search Right side.

② else if (target < arr[mid])
↳ end = mid-1; → left search

③ else if (target > arr[mid])
↳ start = mid+1; → Right search

Inbuilt Function for first Occurance or Last Occurance.

include <algorithm> → std:: lower_bound, upper_bound, sort.

```
auto ans = lower_bound (v.begin(), v.end(), target);
cout << "ans is " << (ans - v.begin()) << endl;
```

Q: Find the total occurance of element in array. (target=4)

2	4	4	4	4	4	4	6	8	10
0	1	2	3	4	5	6	7	8	9

total occurance = 6.

Sol: ukt,

firstOcc = 1

lastOcc = 6

totalOcc = lastOcc - firstOcc + 1
= 6 - 1 + 1
⇒ 6 //

↳ we can apply this because array is sorted.

Q) Find the missing element in array.

(1 to n)

1	2	3	4	6	7	8
0	1	2	3	4	5	6

i=

Sol: observe : index + 1 = element → element − index = 1

if not,
return (index+1);
break;

but how to do this using Binary Search?

Bruteforce sol^n :- sum 1 = 1+2+3+4+5+6+7+8

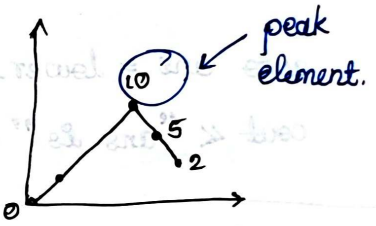
sum 2 = sum of elements of array.

∴ missing element = sum1 − sum2.

Leetcode → Index

(852) Q) Peak Element in a Mountain Array.

0	10	5	2
0	1	2	3



Sol: req^d output = 10 ;

Bruteforce : Linear search → max. element. ⇒ T.S = ≅ O(n)

BS → O(log n)

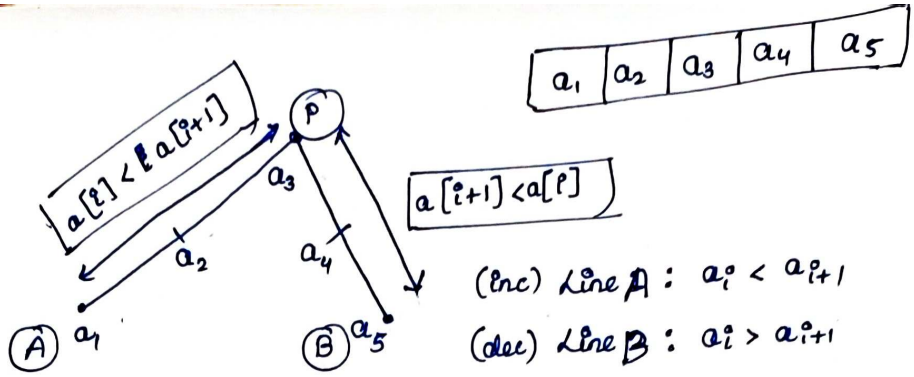
observe : Inc. line is sorted & also dec. line is sorted.

↓ Line 1 ↓ Line 2.

0	10	5	2

s=0 ↑ mid e=3

Sol:



(Inc) Line A : $a_i < a_{i+1}$

(dec) Line B : $a_i > a_{i+1}$

Line (A) → arr [i] < arr [i+1]

⟹ Peak element (P) :→ arr [i-1] < arr [i] > arr [i+1]

Line (B) → arr [i] > arr [i+1]

Conditions :

(1) if (arr [mid] > arr [mid+1])

Possibilities :-

↘ mid element may be the peak element

↳ mid element is in line B.

(2) if (arr [mid] < arr [mid+1]) → mid element is in line A.

↳ NOTE : arr [mid] is NOT a peak element, because peak element kisi se chota nahi hota hai, so check on RIGHT

∴ s = mid + 1;

(3) ↳ else, e = mid;