Today's content

A. **Introduction**

B. **Search for an element K**

C. **search first and last occurrence**

D. **Single element in a sorted Array**

E. **Peak element**

F. **Local minima**

5 m → 3 days.

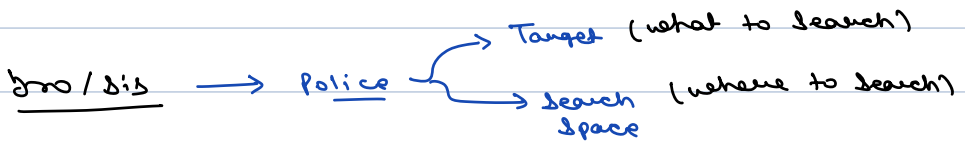
Additional

Mock Interview → DSA.

7 Addi:

↳ DSA ends.

Search story

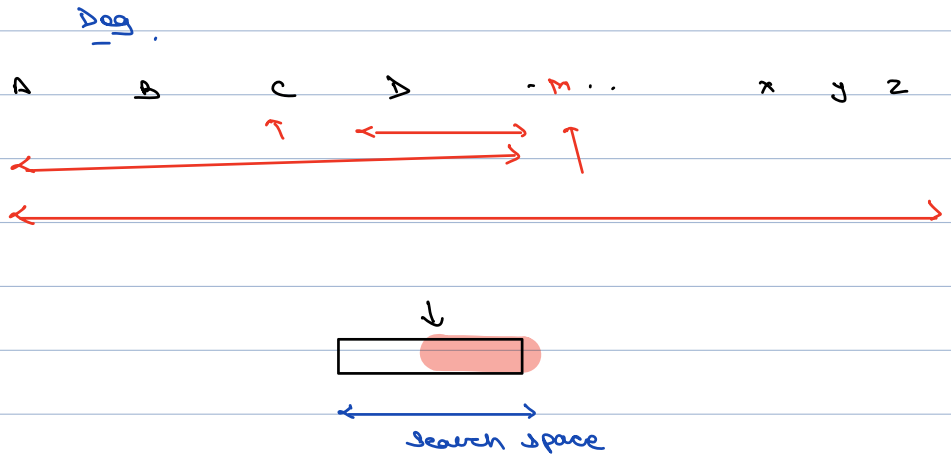


Example

word → {Dict, Book, newspaper}

phone no → phone directory / diary.

Search space is sorted, so searching becomes easy.



Binary Search :-

1) Target
2) Search space
3) Some condn to discard one half of search space.

Binary Search → Divide search space into, 2 parts & repeatedly keep on neglecting one half of the search space.

Ques

Given a sorted array with distinct elements search for index of an element, K, if K is not present return -1.

k = 14 → 4 | k = 99 → -1

	0	1	2	3	4	5	6	7	8	9
arr[10] =	3	6	9	12	14	19	20	23	25	27

idea :-

Linear Search

T.C → O(n)

idea → Binary Search

Target → k

Search Space → array.

① arr[mid] == k

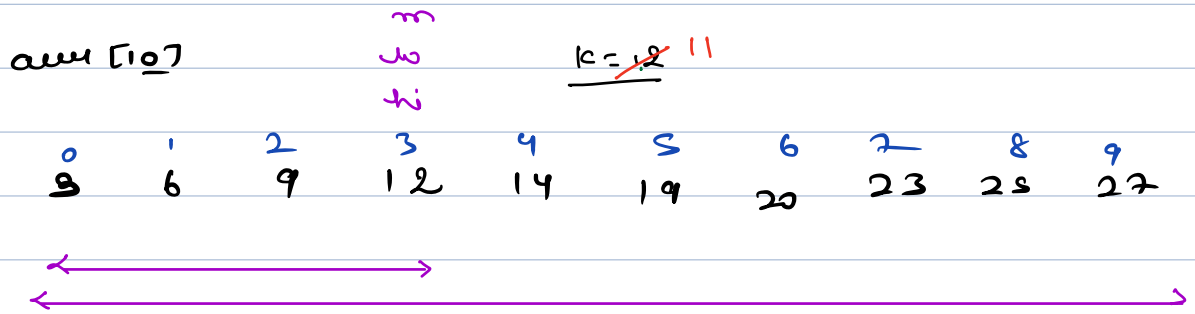
return mid

② arr[mid] < k

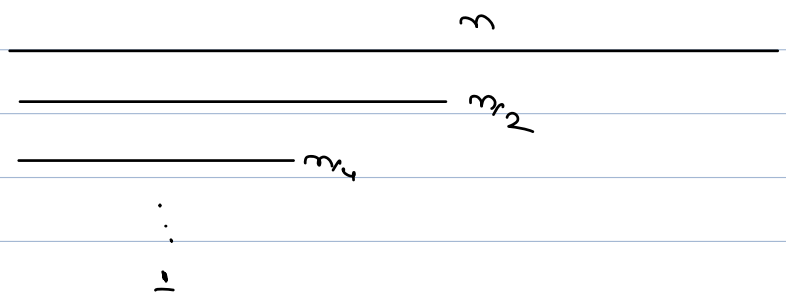
goto right

③ arr[mid] > k

goto left.



lo	hi	mid	
0	9	4	goto left hi = mid-1
0	3	1	goto right lo = mid+1
2	3	2	goto right lo = mid+1
3	3	3	return mid; (break).
3	2		(break the while)



```
int search (int arr[], int n, int k) {
    lo = 0,  hi = n-1
    while (lo <= hi) {
        m = (lo + hi) / 2          → m = lo + (hi - lo) / 2
        if (arr[mid] == k) { return mid }
        else if (arr[mid] < k) { lo = mid+1 }
        else { hi = mid-1 }
    }
    return -1;
}
```

T.C → O(log n)

S.C → O(1)

$m = \frac{lo + hi}{2}$ ✗ $\Rightarrow$ $m = lo + \frac{(hi - lo)}{2}$ $\Rightarrow$ $\frac{2lo + hi - lo}{2}$

$\Rightarrow \frac{lo + hi}{2}$ ✓

int → ∞ → 100

lo = 98, hi = 99

$m = \frac{lo + hi}{2} \Rightarrow \frac{(98 + 99)}{2}$ → Overflow.

2) $m = lo + \frac{(hi - lo)}{2} = 98 + \frac{(99 - 98)}{2} \Rightarrow$

$98 + \frac{1}{2} \Rightarrow 98$.

Ques

Given a sorted array of n element, find first occurrence, index of given element K.

k = 5

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
-5	-5	-3	0	0	1	1	5	5	5	5	5	5	5	8	10	10	15	15

idea 1:-
Linear Search
T.C → O(n).

idea 2
Binary Search
① Target → first occurrence of k
② Search Space → array.

① arr[mid] == k
ans = mid;
go to left,
hi = mid-1;

② arr[mid] < k
goto right
lo = mid+1

③ arr[mid] > k
goto left
hi = mid-1

k = 5

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
-5	-5	-3	0	0	1	1	5	5	5	5	5	5	5	8	10	10	15	15

lo = 7, hi = 8

lo	hi	mid	
0	18	9	ans = 9, goto left, hi = mid-1
0	8	4	move right, lo = mid+1
5	8	6	move right, lo = mid+1
7	8	7	ans = 7, move left, hi = mid-1
7	6	break.	

Todo :- Last occurence

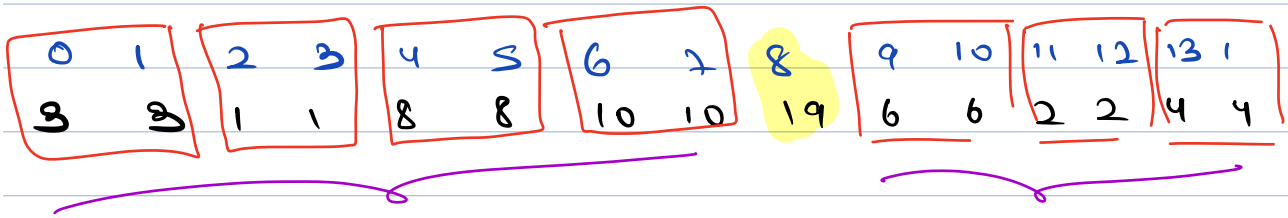
Ques

Given an array where every element occurs twice, except for one unique element, find that unique element

Note:- duplicate elements are adjacent to each other

idea 1 :- use XOR T.C → O(n), S.C → O(1).

Eg :-



first occurence is at even idx (goto right)

first occurence is at odd idx (goto left)

① Target :- unique element

② Search space → arr.

③ Discarding condn.

Tracing.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	3	1	1	8	8	10	10	19	6	6	2	2	4	4

lo	hi	mid	is unique	arr[m] == arr[m-1]	m = m/2
0	14	7	*	m = m-1 = 6	go to right by lo = mid+2
8	14	11	*	*	go to left hi = m-1
8	10	9	*	*	go to left hi = m-1
8	8	8	return unique.		

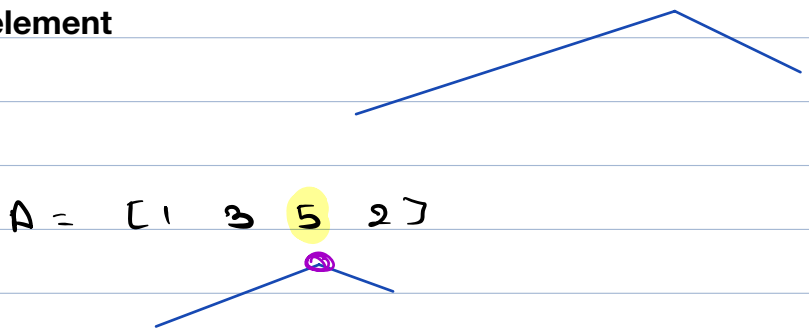
Pseudocode :-

T.C → O(log n)

S.C → O(1).

```
- find Unique ( int arr[], int n) {
    lo = 0, hi = n-1;
    if (n==1) { return 0 }
    if (arr[0] != arr[1]) { return 0 }
    if (arr[n-1] != arr[n-2]) { return n-1 }
    while (lo <= hi) {
        m = lo + (hi-lo)/2
        if (arr[m] != arr[m+1] &&
            arr[m] != arr[m-1]) {
            return m;
        }
        if (arr[m] == arr[m-1]) {
            m = m-1;
        }
        if (mid%2 == 0) {
            lo = m+2;
        }
        else {
            hi = m-1;
        }
    }
}
```

Question: given an increasing decreasing ~~area~~ array with distinct elements, find maximum element



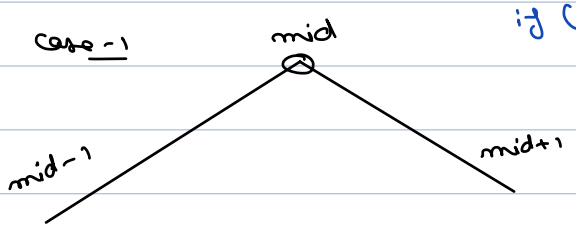
A = [1 3 5 2]

A = [1 3 5 10 15 12 6]

① Target → Peak / max element.

② Search Space → array.

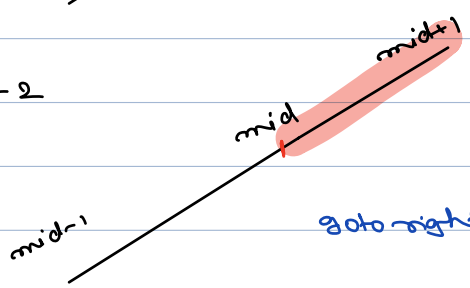
③ Case-1



if (arr[mid] > arr[mid-1] && arr[mid] > arr[mid+1]) return mid

3

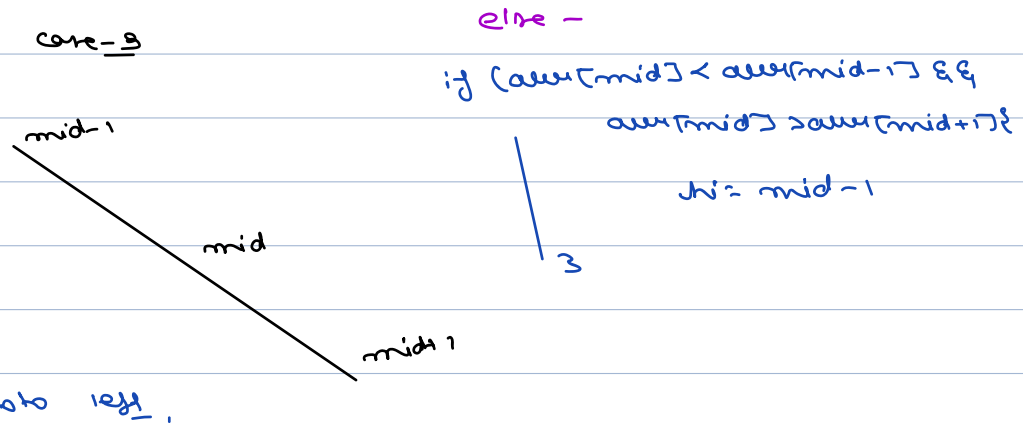
Case-2



if (arr[mid] > arr[mid-1] && arr[mid] < arr[mid+1])

lo = mid+1

3



goto left.

Edge case

① 1 3 5 10

② 15 12 6 2

③ (5)

Question: given an array of N distinct elements, find any local minimum in the array.

minima

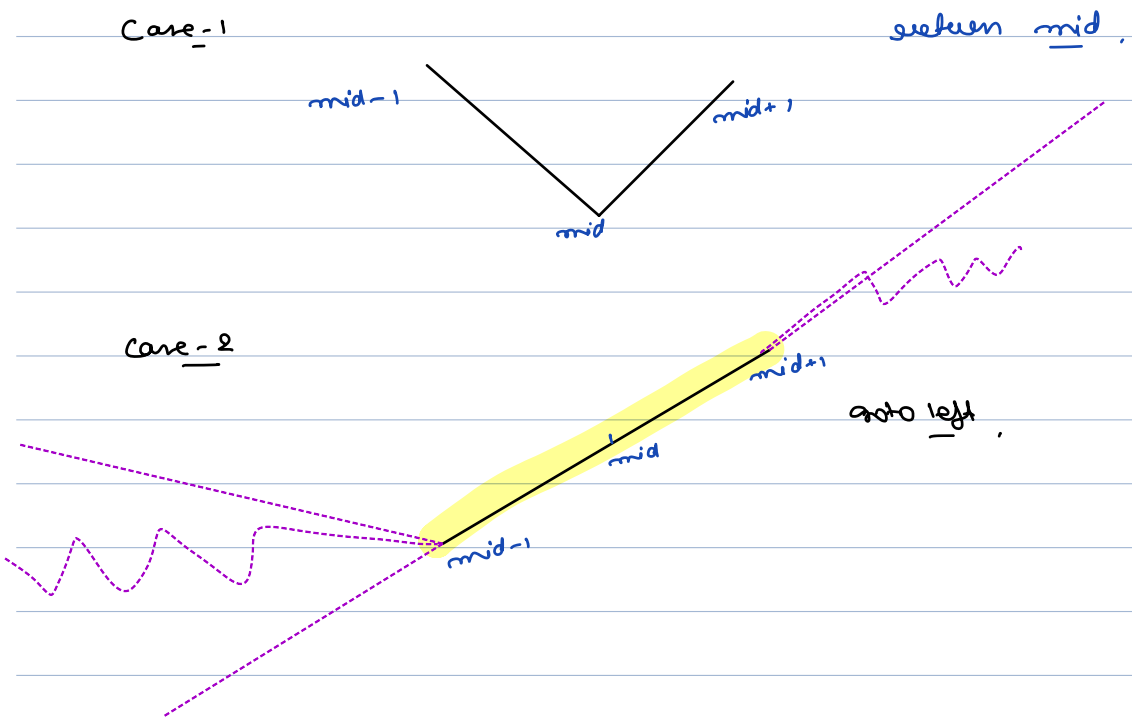
Note: Local ~~minimise~~ a number which is smaller than its adjacent neighbours

A:- 3 6 1 0 9 15 8

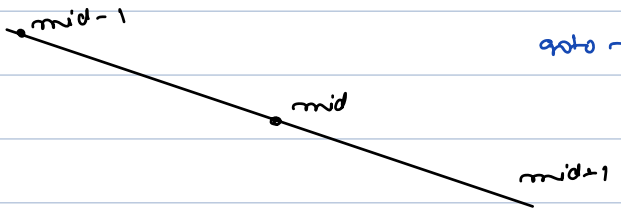
B:- 21 20 19 17 15 9 7.

C:- 5 9 15 16 20 21

d:- 5 8 12 3

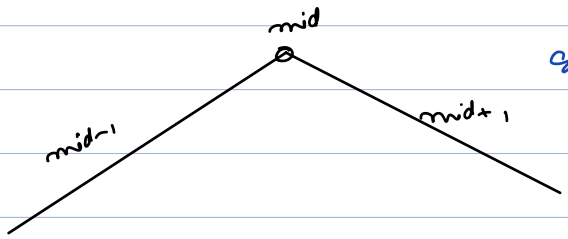


Case-3



go right.

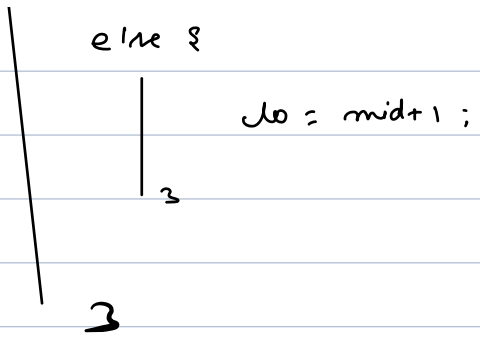
Case-4



go anywhere.

T.C → O(log n)
S.C → O(1)

```
int localMinima (A[]) {
    l = 0,   h = n-1
    if (n==-1) { return arr[0] }
    if (arr[0] < arr[1]) { return arr[0] }
    if (arr[n-1] < arr[n-2]) { return arr[n-1] }
    while (l <= h) {
        m = l + (h-l)/2
        if ( arr[mid] < arr[mid-1] &&
             arr[m] < arr[m+1] ) {
            return mid;
        }
        else if ( arr[mid] < arr[mid+1] ) {
            h = mid-1;
        }
```



idx →	0	1	2	3	4	5	6	7
arr →	9	8	2	7	6	4	1	5

m
hi

l	h	m	A[mid-1]	A[mid]	A[mid+1]	
0	7	3	2	7	6	
4	7	5	6	4	1	goto right
6	7	6	4	1	5