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**

Search story

→ bro / sis → Police → Target (what to search)
→ Search space (where to search)

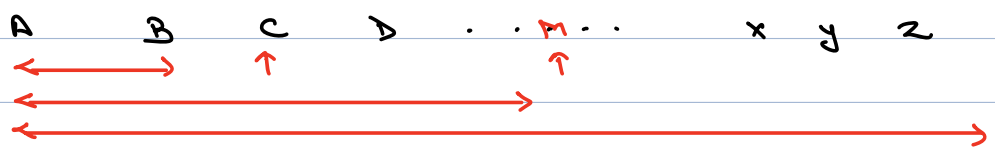
Example

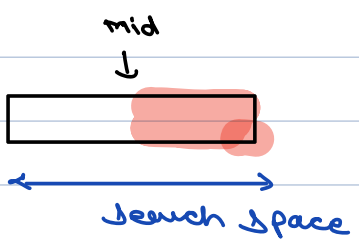
word → {Dict, Book, newspaper}

phone no → phone directory / diary.

Search space is sorted, so searching becomes easy.

→ Dog.





Binary Search :-

1) Target
2) Search Space
3) Some condition to discard one half of the 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.

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

idea1 :-
linear search
↓
T.C → O(n)

idea2 → Binary Search

Target → K
Search Space → array.

① arr[mid] == k

return mid

② arr[mid] < k

goto right

lo = mid+1

③ arr[mid] > k

goto left.

hi = mid-1

arr[10]

K = 12, 11

0	1	2	3	4	5	6	7	8	9
3	6	9	12	14	19	20	23	28	27

m
lo
hi

lo	hi	mid	
0	9	4	hi = mid-1
0	3	1	lo = mid+1
2	3	2	lo = mid+1
3	3	3	return mid;
3	2		break (for ii)

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

mid = lo + (hi-lo)/2

T.C → O(log n)

S.C → O(1)

int → ∞ → 100.

lo = 98, hi = 99

① $mid = \frac{lo+hi}{2} = \frac{(98+99)}{2} =$ overflow

$\Rightarrow \frac{2lo+hi-lo}{2} \Rightarrow \frac{lo+hi}{2}$

② $mid = lo + \left(\frac{hi-lo}{2}\right) \Rightarrow 98 + \frac{(99-98)}{2} \Rightarrow 98 + \frac{1}{2} \Rightarrow 98.$

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
O(n)

idea 2 → Binary Search

Target → first occurrence of K
Search Space → array.

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

② arr[mid] < K
goto right
lo = mid+1

③ arr[mid] > K
goto left
hi = mid-1

k = 5

m
lo hi

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

l	h	mi	
0	18	9	ans = 9, goto left hi = m-1
0	8	4	goto right, lo = m+1
5	8	6	goto right, lo = m+1
7	8	7	ans = 7, goto left, hi = m-1
7	6		Break;

Todo :- Try for last occurence

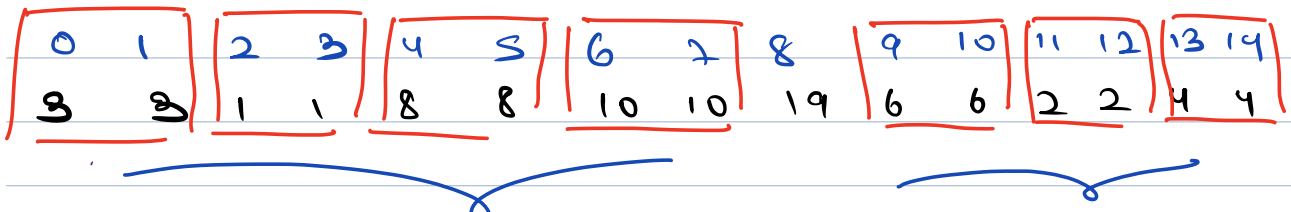
Ques

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).

Ex :-



1) Search Space → arr
2) Target → unique Element.
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

m th lo

lo	hi	mid	is unique	arr[m] == arr[m-1]	m%2
0	14	7	*	m = m-1 6	goto right l = m+2
8	14	11	*	—	goto left hi = m-1
8	10	9	*	—	goto left hi = m-1
8	8	8	True		

Pseudocode :-

T.C → O(log n)

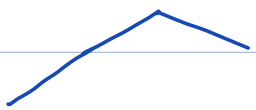
S.C → O(1)

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

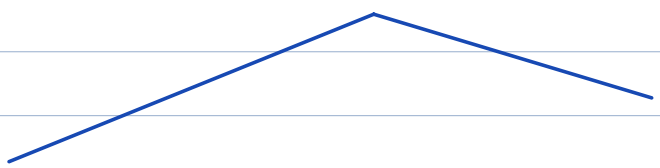
array

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]

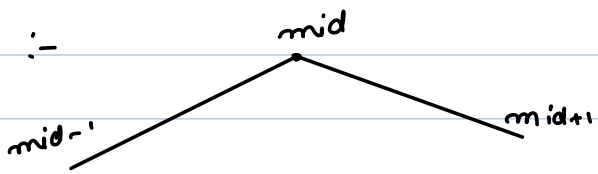


① Target → max element

② Search Space → array.

③

Case-1 :-




```
if (arr[mid] > arr[mid-1]
    && arr[mid] >
        arr[mid+1])
{   return mid;
}
```

Case-2 :-


```
if (arr[mid] > arr[mid-1]
    && arr[mid] <
        arr[mid+1])
{   goto right.
}
```

Case - 3 :-

else {

goto left ;

3

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

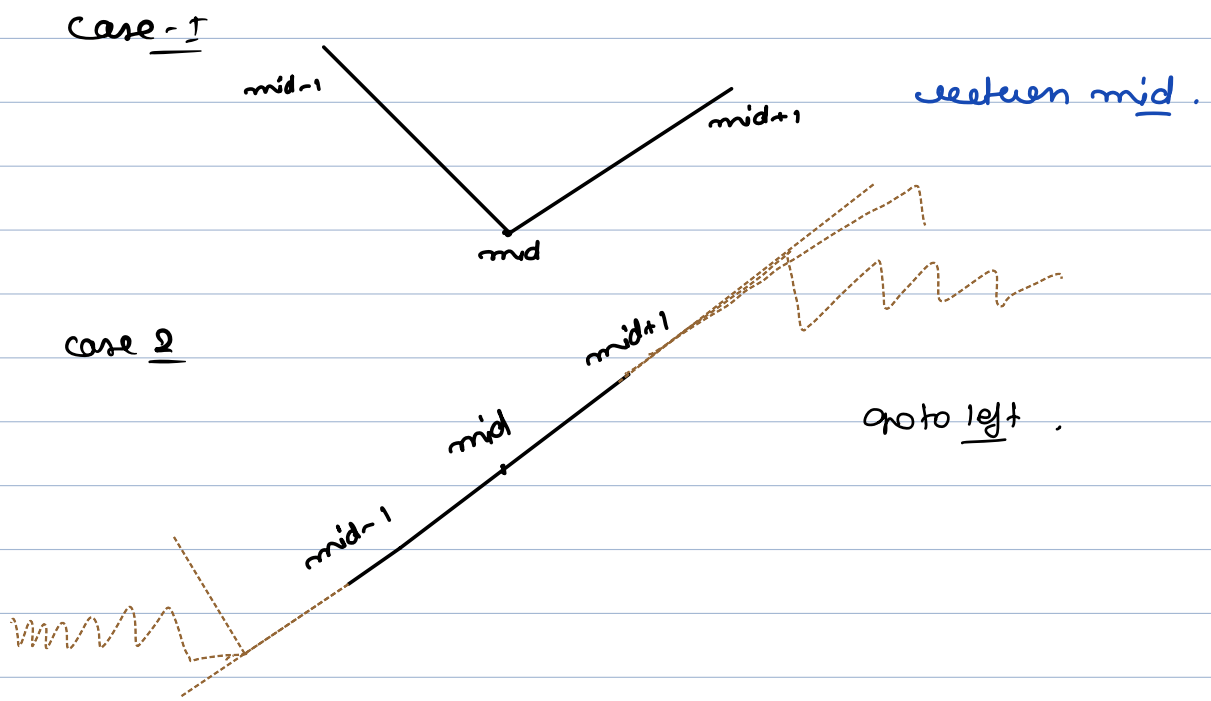
Note: Local ~~minimise~~ a number which is smaller than its adjacent neighbours

A:- 3 6 1 0 9 15 8

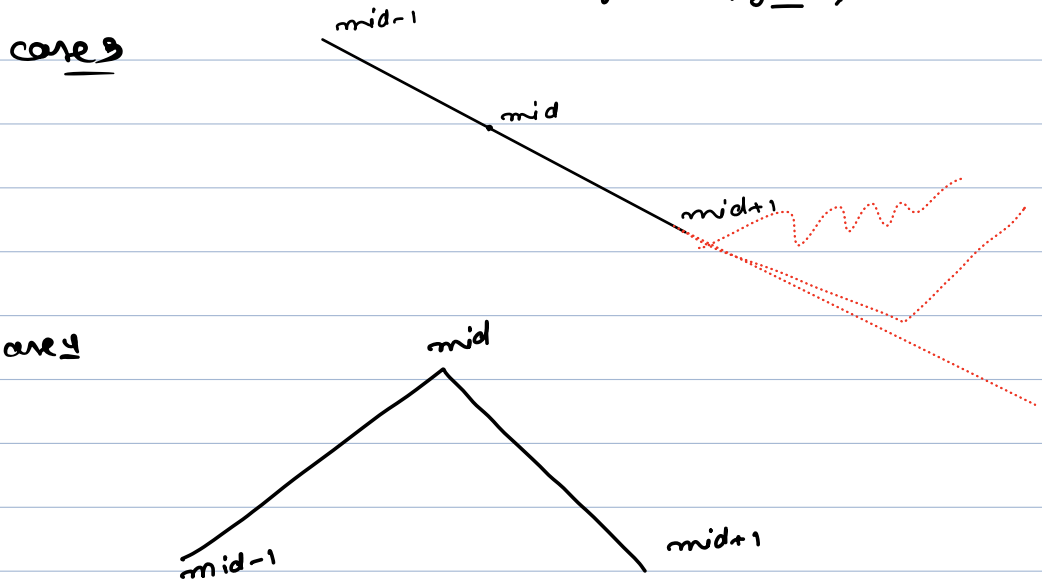
B:- 21 20 19 17 15 9 7.

C:- 5 9 15 16 20 21

d:- 5 8 12 3



goto right.



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[m-1] &&
            arr[m] < arr[m+1]) {
            return mid;
        }
        else if (arr[mid] < arr[mid+1]) {
            hi = mid-1;
        }
    }
```

```
else {
    lo = mid+1;
}
3
```

3

DRY RUN :-

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

m

j h

l	hi	mid	A[mid-1]	A[mid]	A[mid+1]
0	7	3	2	7	6 (goto right)
4	7	5	6	4	1 (goto right)
6	7	6	4	1	5

return :-