BINARY SEARCH

> "Magic is believing in yourself. If you can make that happen, you can make anything happen."
>
> Johann Wolfgang Von Goethe

Today's content

- → Searching Basics
- → Why mid at half?

Problems

- → Search in sorted arr
- → first occurrence of ele in sorted arr
- → Peak element
- → Finding local minima
- → Finding unique element

Searching story –

Target

Bro/Sis ⟶ Police → Details/Photo (what to search)

Police → last location (where to search)

search space

Example

Target — Search space

Word ⟶ { Dict / Newspaper/Books }

phone number ⟶ { Contacts list / Phone diary }

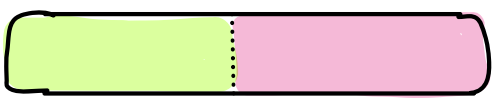
Obs → If search space is sorted, searching becomes easy

Search dog in dictionary

Dict :- [A B C D E F G H I J K . . . . X Y Z]

3 4 2 1

Why land at mid position?



N/2 N/2

Worst → N/2 elements will be discard

Better

N/3 2N/3

Worst → N/3 elements will be discard

When to apply Binary search?

→ After dividing the search space in two halves. if you discard one half of the space using some condition then you can apply BS.

Note :- If you can't discard any search space, then you can't apply Binary search.

Q1. Given a sorted ar[N], search if K is present or not?

ar[10] : { 3 6 9 12 14 19 20 23 25 27 } K=12

0 1 2 3 4 5 6 7 8 9

Idea 1 :- Linear search on arr

TC : O(n)

SC : O(1)

Idea 2 :- Binary search (K)

Case I

ar[mid] == K return True

Case II

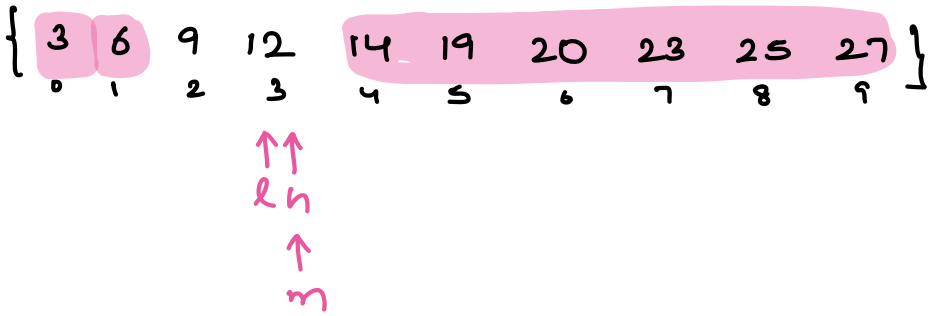
ar[m] > k : discard right

search on left

Case III

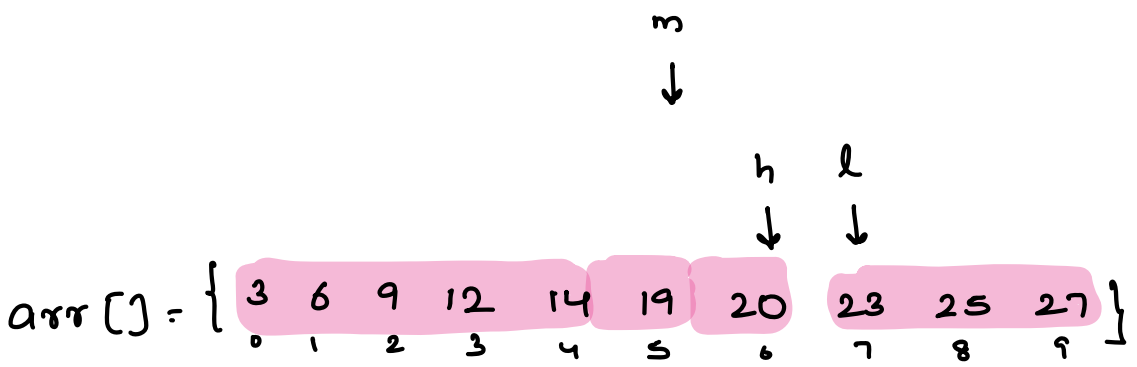
ar[m] < k : discard left

search on right



K = 12

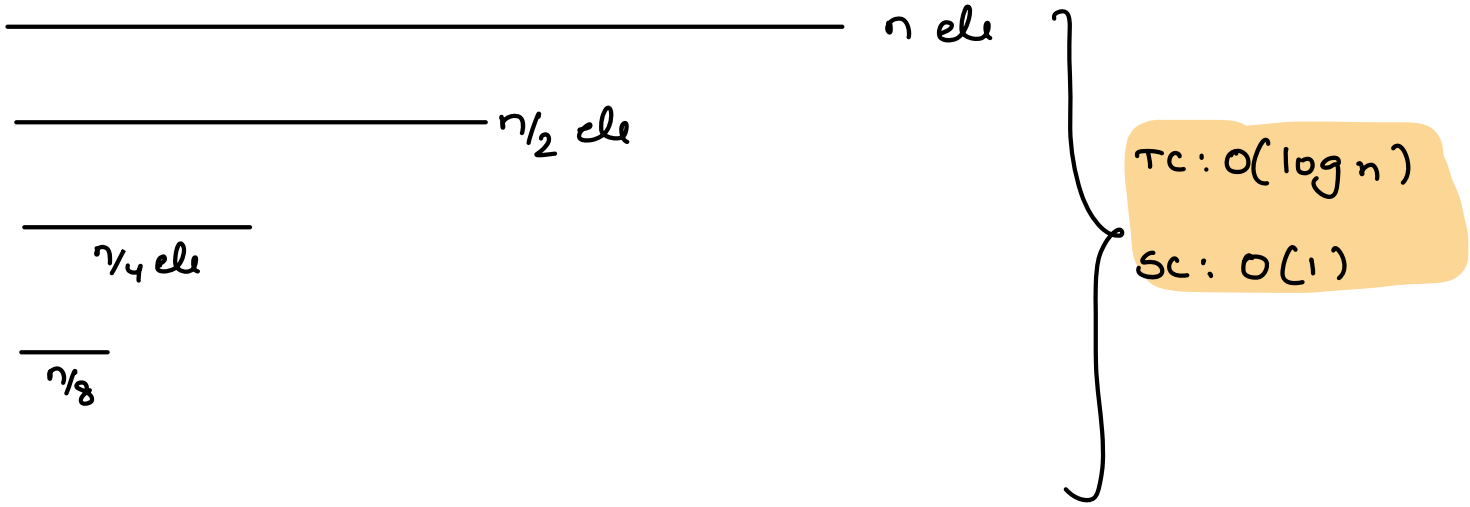
l	h	mid	
0	9	4	ar[4] > 12 : go to left : h = m - 1
0	3	1	ar[1] < 12 : go to right : l = m + 1
2	3	2	ar[2] < 12 : go to right : l = m + 1
3	3	3	ar[3] == 12 : return True



K = 21

l	h	mid		
0	9	4	ar[4] < 21	: l = mid+1
5	9	7	ar[7] > 21	: h = mid−1
5	6	5	ar[5] < 21	: l = mid+1
6	6	6	ar[6] < 21	: l = mid+1
7	6	→	if(l > h) → exhausted the search space	

```
boolean search (int[] A, K)
    int l=0   h=n-1
    while (l ≤ h){
        mid = (l+h)/2
        if (A[m]==K) return true:
        else if (A[m]>K) h=m-1
        else l=m+1
    }
return false
}
```



Q - Given a **sorted array** of elements . find **first index** of a given target

A[] =

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

K	first idx
-5	0
5	7
8	14

Idea 1 → Linear search on arr

TC : O(n)

SC : O(1)

Idea 2 Binary Search

Case I

ar[m] > K : h = m-1

moving on left

ar[m] < k : l = m + 1

move on RHS

(ar[m] == k)

ans = mid ;

h = mid - 1

```
int firstIndex ( int [] A, K )
    int l = 0   h = n-1 . ans = -1
    while (l ≤ h){
        mid = (l+h)/2
        if ( A[m] == k){
            ans = m;
            h = m-1
        }
        else if (A[m] > k)  h = m-1
        else  l = m+1
    }
    return ans ;
}
```

TC: O(log n)

SC: O(1)

Search for last index : {TODO}

03. Given an increasing decreasing array with distinct elements Find peak element

arr [ ] = { 1 3 5 2 }

arr [ ] = { 1 3 5 10 15 12 6 }

ar [ ] = { 5 }

ar [ ] = { 1 2 6 10 }

ar [ ] = { 10 2 6 1 }

Idea 1 → Sort the entire array & $(n-1)^{th}$ idx is ans

TC : O(nlogn)

SC : O(1)

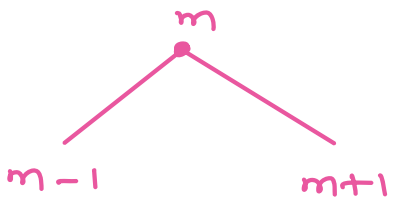
Idea 2 → Iterate & search for max ele

TC : O(n)

SC : O(1)

Idea3 → Use Binary search

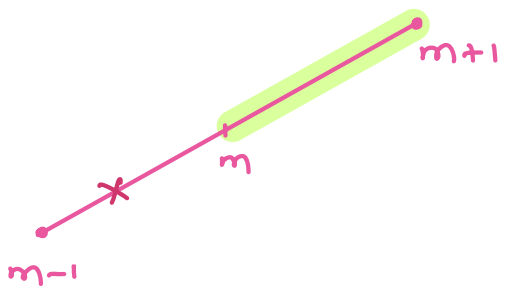
Case I



→ if (A[m] > A[m-1] &&
A[m] > A[m+1])

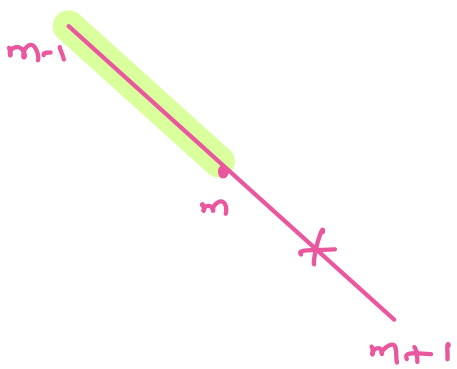
return A[m]

Case II



→ if (A[m] < A[m+1] &&
A[m] > A[m-1]) go to RHS

Case III



if (A[m] > A[m+1] &&
A[m] < A[m-1]) go to LHS

```
if ( A.length == 1) return A[0]

if (A[0] > A[1] )  return A[0]

if (A[n-1] > A[n-2]) return A[n-1];

  lo = 1   , hi = n-2

  while (lo ≤ hi)
  |
  |    mid = (lo+hi) / 2
  |
  |  if (A[m] > A[m-1] && A[m] > A[m+1])
  |  |
  |  |   return A[m] ;
  |  3
  |  else if (A[m] < A[m+1] && A[m] > A[m-1])
  |  |
  |  |    l = m+1 ;
  |  3
  |  else {
  |  |
  |  |    h = m-1 ;
  |  3
  |
  3
```

TC : O(lgn)

SC : O(1)

10:05 → 10:15 pm

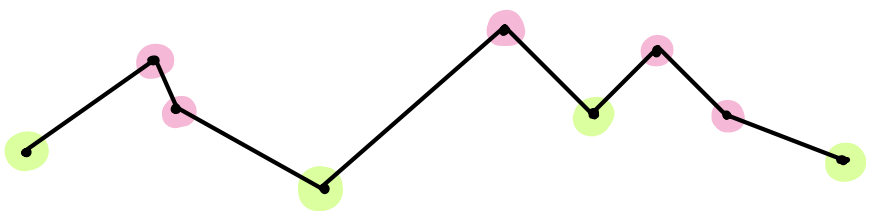
04. Given unsorted arr[] distinct elements, return any local minima. An element is said to be local minima, if it is less than its adjacent elements.

Eg:- arr[8] = { 9 8 7 3 6 4 1 5 2 }
0 1 2 3 4 5 6 7 8

Local minima → arr[i-1] > arr[i] < arr[i+1]

arr[0] < arr[1]

arr[n-1] < arr[n-2]



arr[] = { 9 8 7 3 6 4 1 5 2 }
0 1 2 3 4 5 6 7 8

Idea 1 → Iterate on array & compare k^{th} idx element

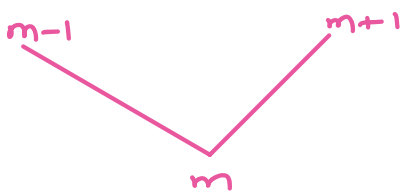
ar[k-1] & ar[k+1]

TC: O(n)

SC: O(1)

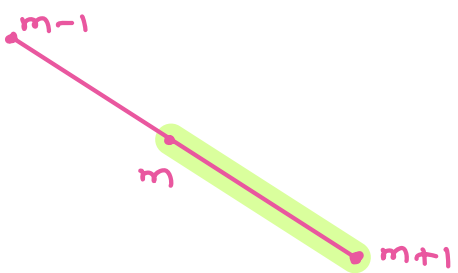
Idea 2 → Use Binary search

01. Case I



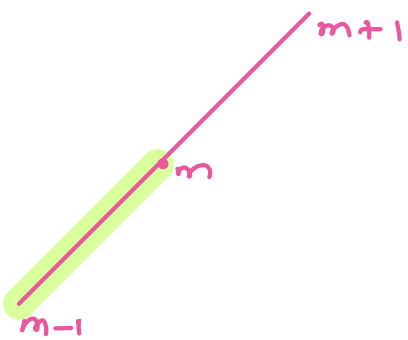
→ return ar[m]

02 Case II



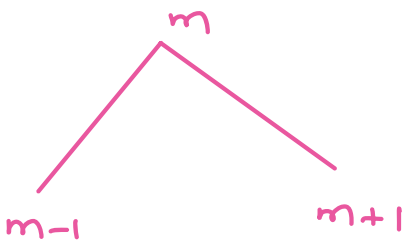
go to RHS

03. Case III



go to LHS

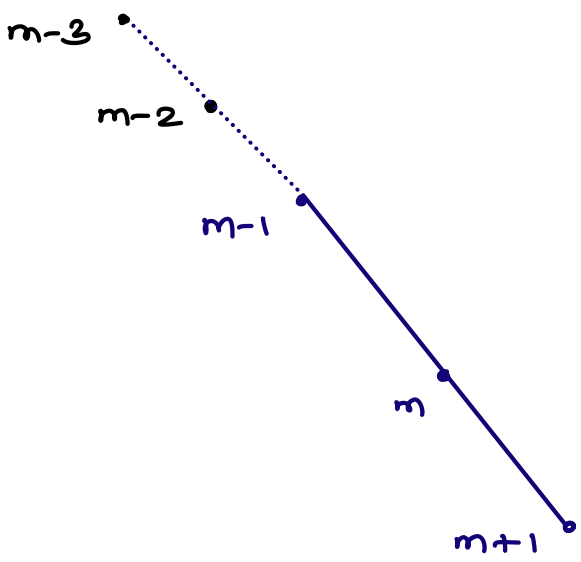
04.



→ go in either direction

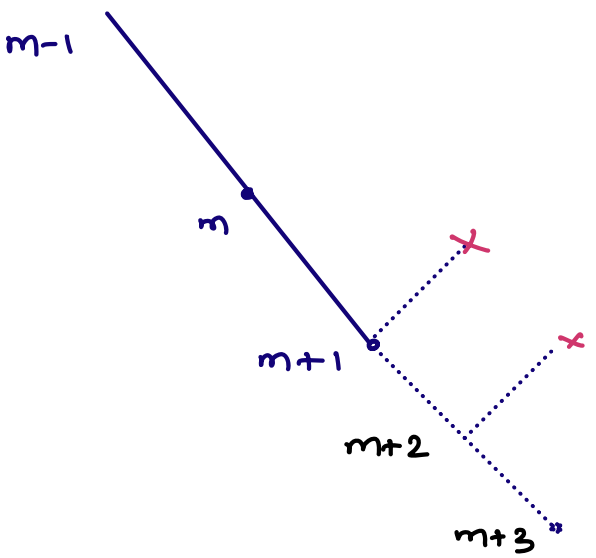
* Case II

(a) Analysing LHS first



Worst case scenario → there is no dip in LHS

* Analyse the RHS



→ Always going to have atleast one answer

```
if (n==1) return A[0]

if (A[0] < A[1]) return A[0]

if (A[n-1] < A[n-2]) return A[n-1]

l=1     h=n-2

while (l ≤ h)
|
|    mid= (l+h)/2
|
|    if ( A[m] < A[m-1]  &&  A[m] < A[m+1]) return A[m]
|
|   else if (A[m] < A[m-1] && A[m] > A[m+1]  l= m+1
|
|   else   h=m-1;
|
3
```

05. Every element in an array occurs twice except for one. Find the unique element.

Note :- Duplicates are adjacent to each other.

arr [ ] =

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

Idea 1 → XOR of all elements

TC: O(n)

SC: O(1)

Idea 2 → Use freq arr to store freq of all distinct ele

TC: O(n)

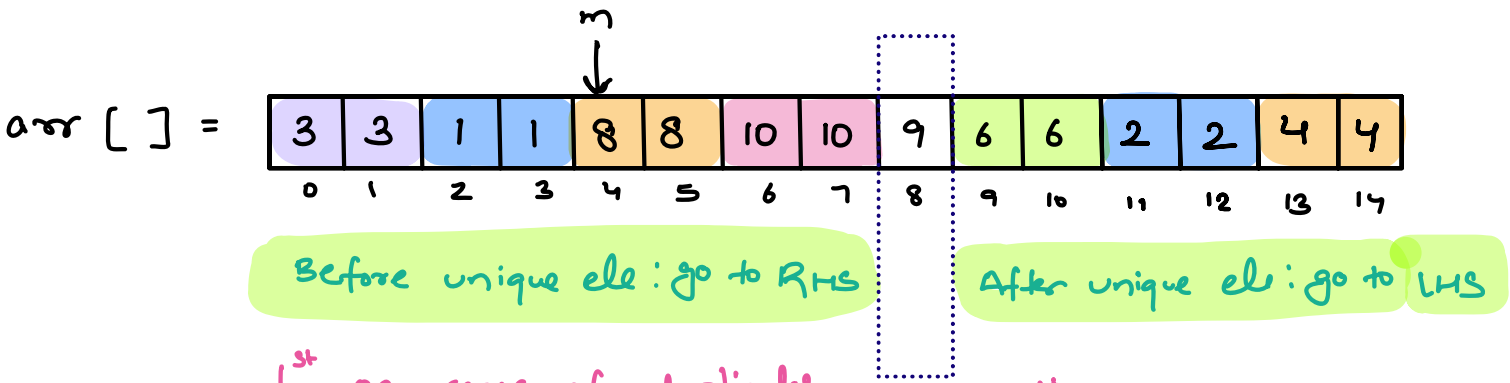
SC: O(n)

Idea 3 → Linear search & check for the adjacent ele

TC: O(n)

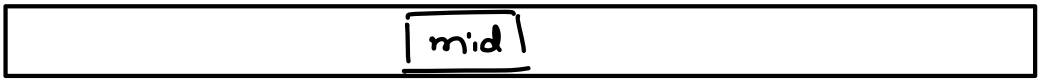
SC: O(1)

Idea 4 → Use Binary Search



1st occurrence of duplicates is even idx

1st occurrence of duplicates is at odd idx



duplicate mid

duplicate mid

1st idx of duplicate

even

go to RHS

ar[m] != ar[m-1] &&

ar[m] != ar[m+1]

return ar[m]

1st idx of duplicate

odd

go to LHS

```
if (n==1) return ar[0]

if (ar[0] != ar[1]) return ar[0]

if (ar[n-1] != ar[n-2]) return ar[n-1]
```

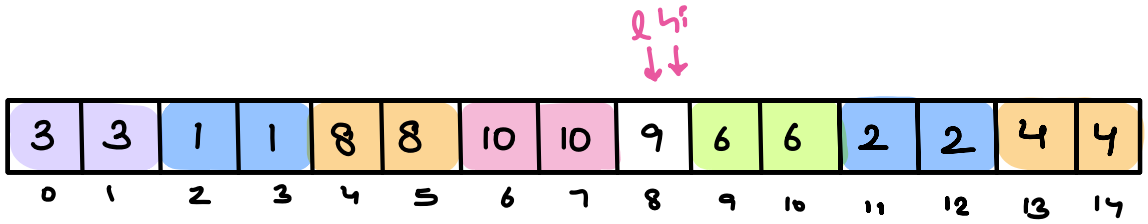
l = 1 h = n-2

```
while (l ≤ h)
{
    mid = (l+h)/2

    if (A[m] != A[m-1] && A[m] != A[m+1])
        return A[m]

    if (A[m-1] == A[m]) {
        m = m-1        // moving mid to 1st idx of duplicates

    if (m % 2 == 0) {
        lo = m+2       go to RHS
    else {
        h = mid-1;
}
```

TC : O(log n)

SC : O(1)

m
↓



l	h	mid
1	13	7
8	13	10
8	8	8

m = 6 : go to RHS l = mid+2

m = 9 : go to LHS h = mid-1

return true :