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

Problems

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

Searching

Bro/Sis Missing → Police → Details/Photos (what to search) — Target

Police → last seen (where to search) — Search space

Example

Word → { Dictionary, Newspaper, Book }

Phone no. → { contact list, Diary, Phonebook }

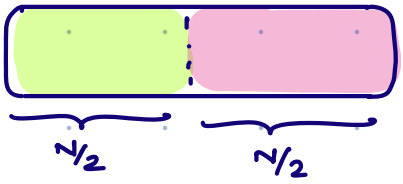
Conclusion → If search space is sorted, searching becomes easy.

* Search "Dog" in dictionary

dict = { A B C D E F G ... Z }

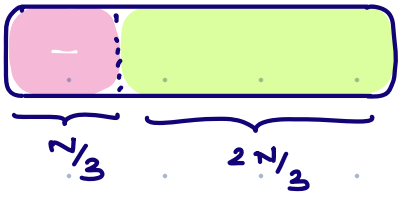
In binary search, at each step, the search range is typicall half

Why to always prefer to land at mid ?



Better

Worst : $N/2$ elements are getting discarded



Worst : $\frac{N}{3}$ elements are getting discarded

When to Apply Binary Search ?

→ After splitting the search space, if you are able to discard one of the search space using some condition then we can go & apply Binary Search.

01. Given an sorted arr[]. search if K is present or not ?

A[] = { 3 6 9 12 14 19 20 23 }

K = 12 → True

K = 16 → false

Brute force → Iterate through array & search for K

TC : O(n)

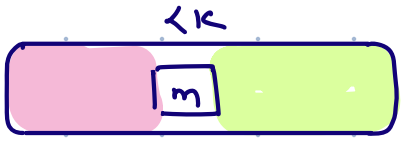
SC : O(1)

Idea 2 Binary Search

Case I

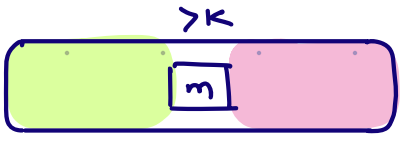
A[mid] == K → return true

Case II



if (A[m] < K) → discard left
→ move towards right

Case III



if (A[m] > K) → discard right
→ move towards left

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

m, l, h

K = 12

Tar = 12

Search space = 0 to 9

Mid = $\frac{(l+h)}{2}$

l	h	mid	compare A[mid] & K
0	9	4	A[4] > 12 : go to LHS, h = mid - 1
0	3	1	A[1] < 12 : go to RHS, l = mid + 1
2	3	2	A[2] < 12 : go to RHS, l = mid + 1
3	3	3	A[3] == 12

Return true :

m
↓
h l
↓ ↓

A[ ] = { 3 6 9 12 14 19 20 23 25 27 } K = 35

0 1 2 3 4 5 6 7 8 9

lo	hi	mid	compare A[mid] & K
0	9	4	A[4] < 35 go to RHS, l = mid+1
5	9	7	A[7] < 35 go to RHS, l = mid+1
8	9	8	A[8] < 35 go to RHS, l = mid+1
9	9	9	A[9] < 35 go to RHS, l = mid+1
10	9	→ search space is exhausted	

```
boolean search ( int [] A, int K)
{
    int lo = 0
    int hi = n-1

    while ( lo ≤ hi ){
        m = (lo + hi)/2

        if (A[m] == k) return true;
        else if (A[m] < k) lo = m+1;
        else hi = m-1;
    }
    return false
}
```

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

TC: O(log n)

SC: O(1)

$N \rightarrow \frac{N}{2} \rightarrow \frac{N}{4} \rightarrow \frac{N}{8} \ldots 1 \Rightarrow \log n$

Let's ponder upon a real life problem -
"All emails in your mailbox are sorted chronologically. Can you find the first mail that you received in 2020?"

Suppose, in an array, we are given year in which an ith email was received, sorted in lexicographical order. Given an year, we need to return the index of first email of that year.

arr[] =

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
2005	2005	2013	2018	2018	2019	2019	2020	2020	2020	2020	2020	2020	2020	2022	2023	2023	2024	2024

NOTE: Please slide to right for checking entire array.

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

A[] =

m

-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

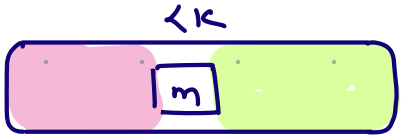
	Ans
K = 5 ⟶	7
K = 10 ⟶	15
K = -5 ⟶	0

Idea 2 Binary Search

Case 1


```
if (A[m] == K) {
    ans = m;
    hi = m - 1
}
```

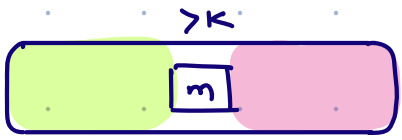
Case II



if (A[m] < K) : goto RHS

l = m + 1

Case III



if (A[m] > K) go to LHS

hi = m - 1

```
int firstIndex (int [] A, int K)
{
    int lo = 0
    int hi = n-1
    ans = -1
    while (lo ≤ hi) {
        m = (lo + hi)/2
        if (A[m] == K) {
            ans = m ; hi = m-1 ;
        }
        else if (A[m] < K) lo = m+1 ;
        else hi = m-1 ;
    }
    return ans ;
}
```

TC: O(log n)

SC: O(1)

* Try for last occurrence as well.

Peak element

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

A[] = {1 3 5 2} Ans = 5

A[] = {1 3 5 6 2} Ans = 6

A[] = {1 2 6} Ans = 6

A[] = {10 2 1} Ans = 10

A[] = {5} Ans = 5

Idea 1 → Iterate & search for maximum element

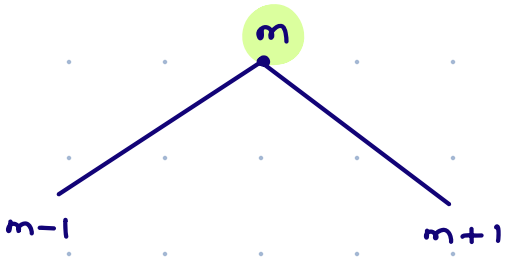
Tc : O(n)
Sc : O(1)

Idea 2 → Sort the entire array & return last element

Tc : O(nlogn)
Sc : O(n)

Idea 3 : Binary search

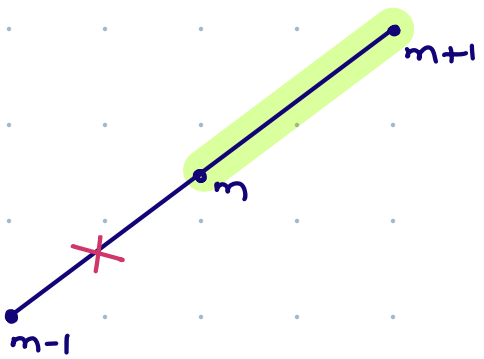
Case 1



→ if (A[m] > A[m-1] && A[m] > A[m+1])

return A[m]

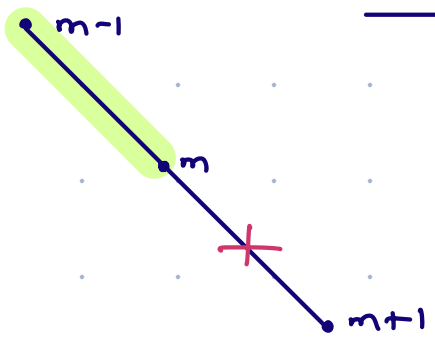
Case II



→ if (A[m] > A[m-1] && A[m] < A[m+1])

: goto RHS

Case III



→ if (A[m] < A[m-1] && A[m] > A[m+1])

: goto LHS

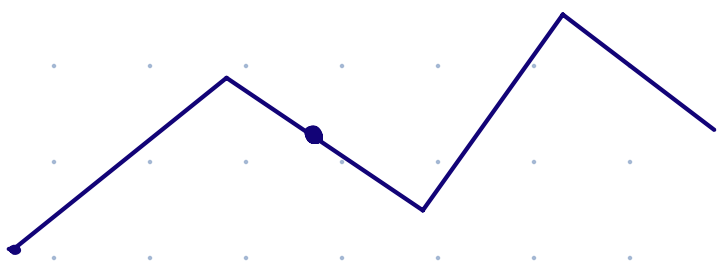
if (A.length == 1) return A[0]

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){

    m = (lo+hi)/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]) lo = m+1;

    else if (A[m] < A[m-1] && A[m] > A[m+1]) hi = m-1

}
```

3 10 7 6 5 14 3 2 1



10:37 pm → 10:47 pm

Local Minima

Given an unsorted arr of distinct elements, return any local minima.

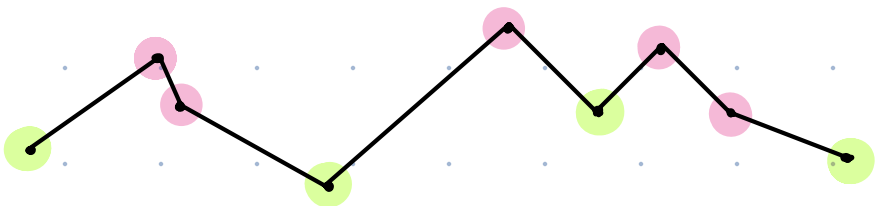
An element is a local minima, if it is less than its adjacent elements.

$$ar[] = \{9\ 8\ 7\ 3\ 6\ 4\ 1\ 5\ 2\}$$

Local minima → $ar[i-1] > ar[i] < ar[i+1]$

$ar[0] < ar[1]$

$ar[n-1] < ar[n-2]$



Idea 1 → Iterate on array & check if

$A[k] < A[k-1]$ &&

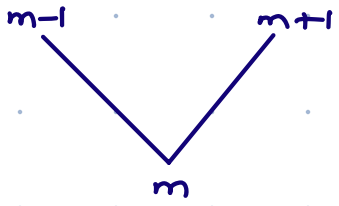
$A[k] < A[k+1]$

TC : O(n)

SC : O(1)

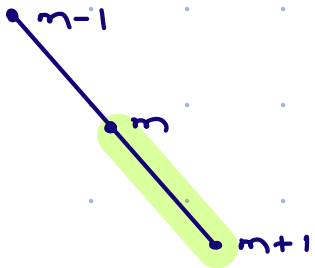
Idea 2 Binary Search

01. Case 1



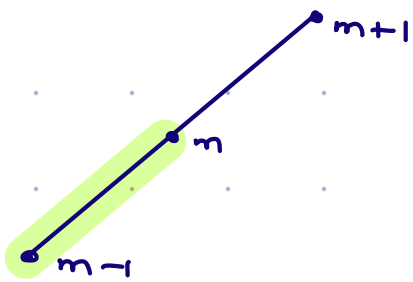
→ return A[m]

02 Case II



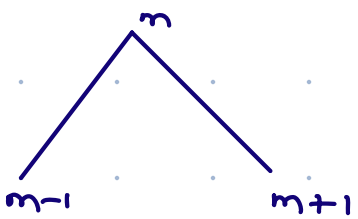
→ move towards RHS

03. Case III



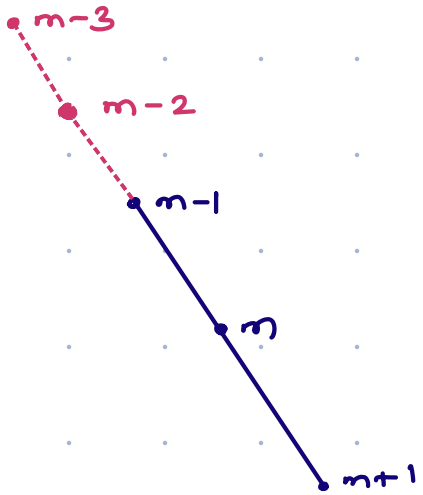
→ move towards LHS

04. Case IV



→ Either side will work

* Analysing the second co e

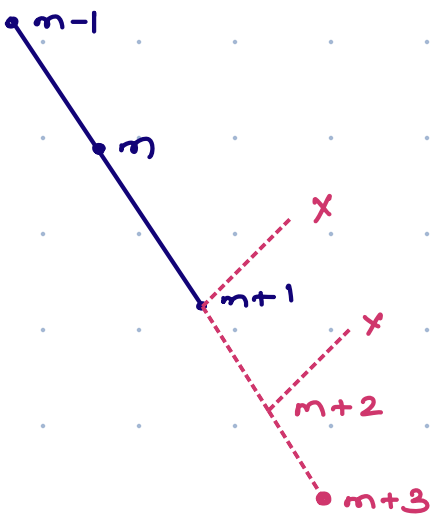


For LHS

Worst case scenario →

There is no dip on LHS & hence no local minima

RHS



Obs → There will always be one local minima present on RHS

```
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]

l = 1
h = n-2

while (l ≤ h) {
    m = (lo + hi) / 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]) lo = m+1
    else hi = m-1;
}
```

Tc : O(log n)
Sc : O(1)

Unique Elements

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

01. Take XOR of all elements

TC : O(n)
SC : O(1)

02. Use hashset

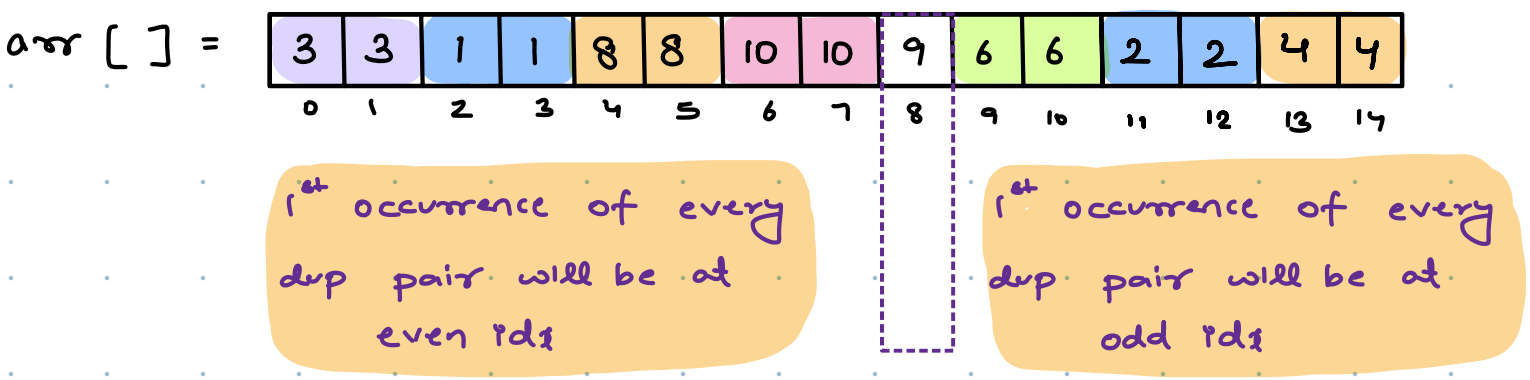
TC : O(n)
SC : O(n)

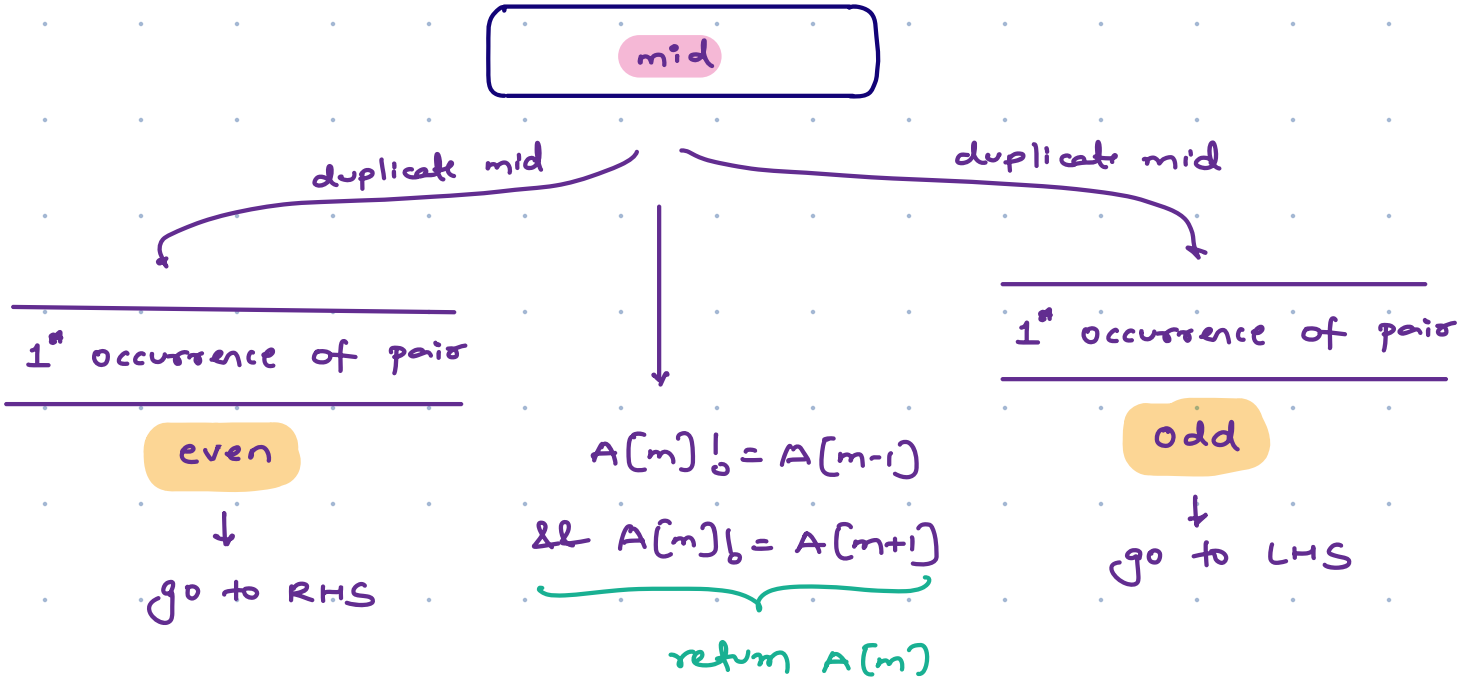
03. Freq Array

TC : O(n)
SC : O(range of ele)

04. Linear search & find the unique element

TC : O(n)
SC : O(1)



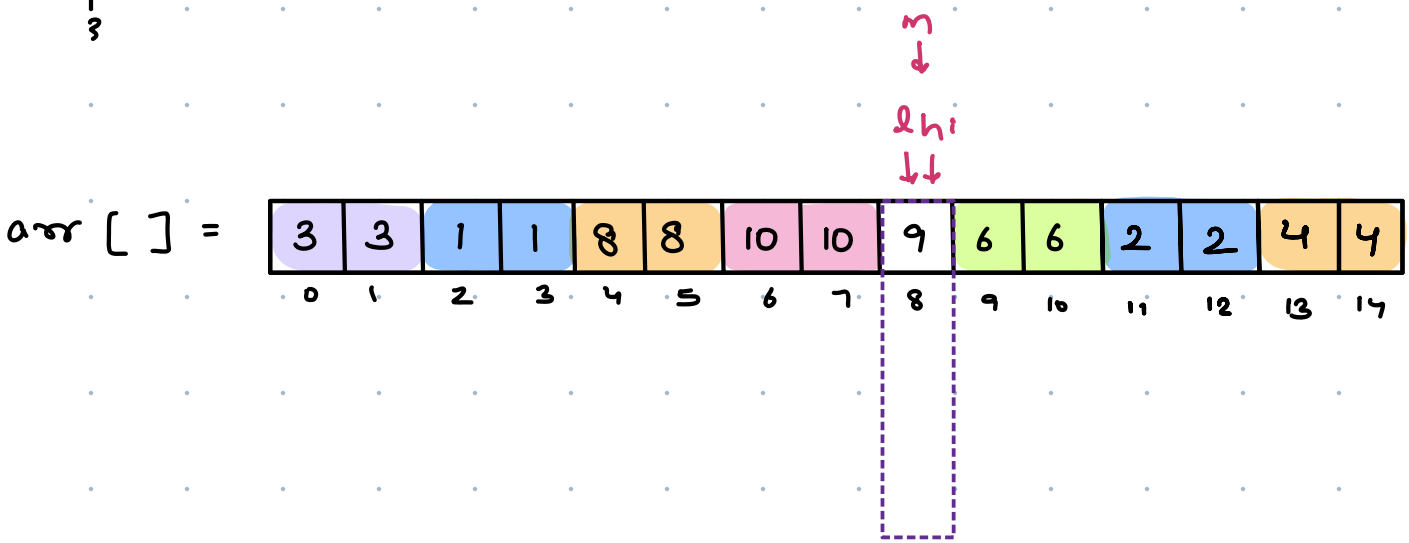



```
if (n == 1) return A[0]

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 (lo <= hi){

    int m = (lo + hi)/2

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

        return A[m];
    }

    if (A[m] == A[m-1]){
        m = m-1;        // move mid to 1st occurrence
    }
```

```
if (m%2==0){
    lo=m+2;
}
else{
    hi=m-1;
}
```

TC: O(log n)

SC: O(1)



lo	hi	mid	
0	14	7	
8	14	11	
8	10	9	
8	8	8	→ return A[8]