Agenda

- What is Binary search ?
- Find an element in array
- Find first occurrence of element in array
- Unique Element
- Local Minima

Introduction to Searching

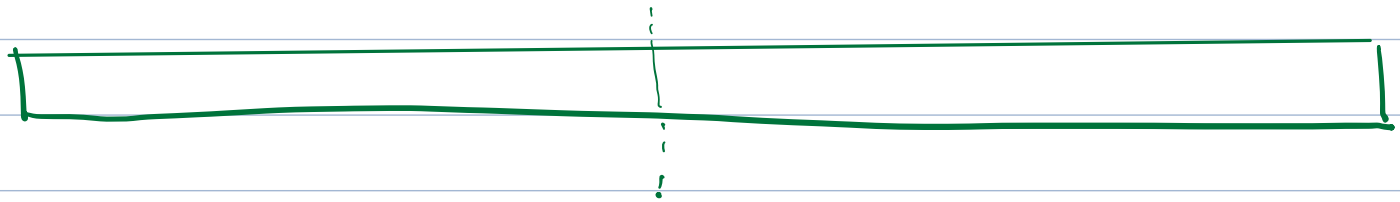
Target - What to search for

Search space - Where to search

Searching in organised search space is easier:

1. Dictionary
2. Phone book

Binary Search



Divide search space into 2 parts and based on some condition, we repeatedly neglect one-half of search space

↓

1. Till we get ans
2. No search space is left

At each step, search space is halved

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

0	1	2	3	4	5	6	7	8	9
3	6	9	12	14	19	20	23	25	27

K = 12 ans = 3

K = 13 ans = -1

Brute Force: Linear search

Iterate entire array to check if k is present

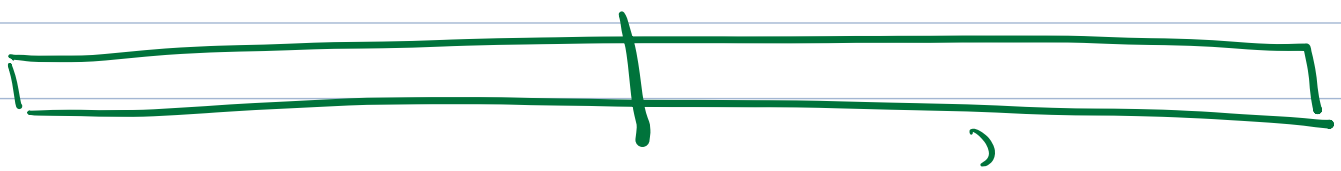
TC: O(N) SC: O(1)

Binary Search

Search space: Array (0 → N-1 idx)

Target: k

Condition:



A[mid] == k

return mid

else if (A[mid] < k)

go to right

else

go to left

0	1	2	3	4	5	6	7	8	9
3	6	9	12	14	19	20	23	25	27

K = 12

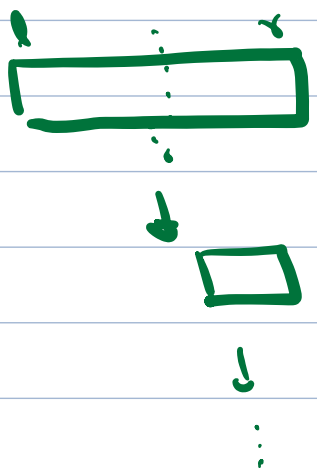
l	r	mid $\frac{l+r}{2}$	A[mid]	where to go
0	9	4	14	14 > 12 Go to left r = mid - 1
0	3	1	6	6 < 12 Go to right l = mid + 1
2	3	2	9	9 < 12 Go to right l = mid + 1
3	3	3	12	12 == 12 return 3

---

l	r	mid		
3	3	3	12	K = 13 12 < 13 Go to right l = mid + 1
4	3			l > r STOP

```
int search(int a[], int N, int K){
    l = 0, r = N-1
    while(l <= r){
        mid = (l+r)/2
        if(a[mid] == K)
            return mid
        else if(a[mid] < K)      // right
            l = mid+1
        else                     // K < a[mid]   // left
            r = mid-1
    }
    return -1
}
```

TC: $O(\log_2 N)$
SC: $O(1)$



N → N/2 → N/4 → ... → 1

Best practice to compute Mid

Let's assume that we have a datatype → dtype which has a range -100 to 100.

Array of length 100 ⇒ 0 to 99 indices

① dtype $l = 0$, $r = 99$

dtype $mid = \frac{l+r}{2}$

⇒ go right $l = mid + 1$

② $l = 50$ $r = 99$

$mid =$

$$Mid = \frac{l+r}{2} = \boxed{l + \frac{(r-l)}{2}}$$ ✓ Mid

$l = 50$ $r = 99$

2. All emails in your mailbox are sorted chronologically. Can you find first email that you received in 2020?

0	1	2	3	4	5	6	7	8
2005	2009	2013	2018	2019	2019	2020	2020	2020

9	10	11	12	13	14
2020	2021	2021	2022	2023	2024

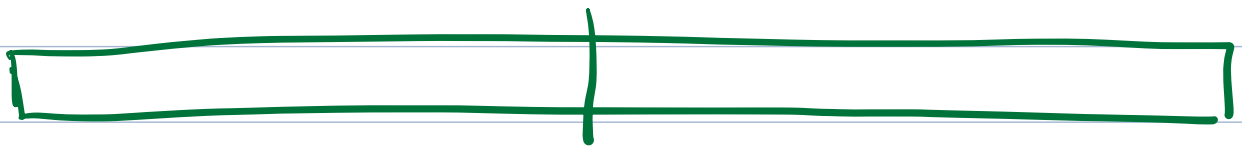
ans = 6 Sorted array of duplicates

Brute Force: Linear search from left to right and return idx / first time you find 2020

TC: O(N) SC: O(1)

Binary Search:

Search space: Array (0 → N-1 idx)
Target: First occurrence of 2020
Condition:



A[mid] == 2020
ans = mid
Search on left
r = mid-1

A[mid] < 2020 Right

l = mid + 1

A[mid] > 2020 Left

r = mid - 1

0	1	2	3	4	5	6	7	8
2005	2009	2013	2018	2019	2019	2020	2020	2020

9	10	11	12	13	14
2020	2021	2021	2022	2023	2024

K = 2020

l	r	mid $\frac{l+r}{2}$	A[mid]	Where to go	ans -1
0	14	7	2020	2020 == 2020 Update ans Left, r = mid - 1	7
0	6	3	2018	2018 < 2020 Right, l = mid + 1	7
4	6	5	2019	2019 < 2020 Right, l = mid + 1	7
6	6	6	2020	2020 == 2020 Update ans Left, r = mid - 1	6

6 5 l > r End of search

ans = 6

```
int search( int a[], int N, int K) {
    l = 0, r = N-1 , ans = -1
    while ( l <= r) {
        mid = (l+r) / 2
        if ( a[mid] == K) {
                ans = mid
                r = mid - 1          // left
        }
        else if ( a[mid] < K)        // right
                l = mid + 1
        else                         // a[mid] > K  -> left
                r = mid - 1
    }
    return ans
}
```

N → N/2 → ... → 1

TC : O(log₂N)

SC : O(1)

Q: Find last occurrence of an element k

Q: Find freq of an element in sorted arr

last occurrence idx - first occurrence idx + 1

10:25

https://notability.com/n/0ObYmqKOh2vA9~zPNOZ7MR

3. Every element occurs twice except for 1, find the unique element

NOTE: Duplicate elements are adjacent to each other but the array is not sorted

	0	1	2	3	4	5	6	
A →	8	8	5	5	6	2	2	ans = 6

BF : ① check every elem A[i] with A[i+1]

i = 0 → 2 → 4 ....

stop when A[i] != A[i+1]

unique elem → A[i]

TC : O(N)

SC : O(1)

② XOR of array

Binary Search

Search space : Array (0 → N-1 idx)

Target : Unique Element

Condition :

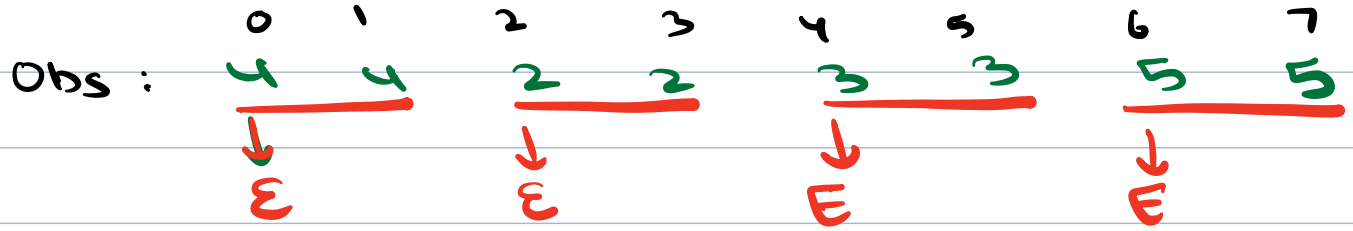
① A[mid] is unique elment

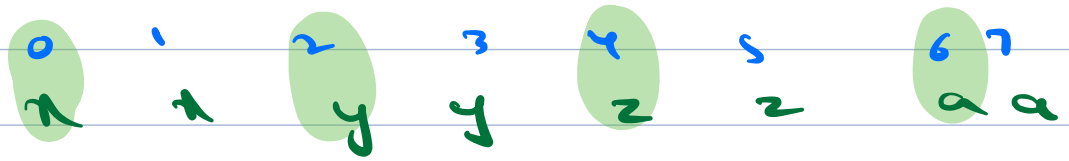
if (A[mid] != A[mid-1] &&

A[mid] != A[mid+1])

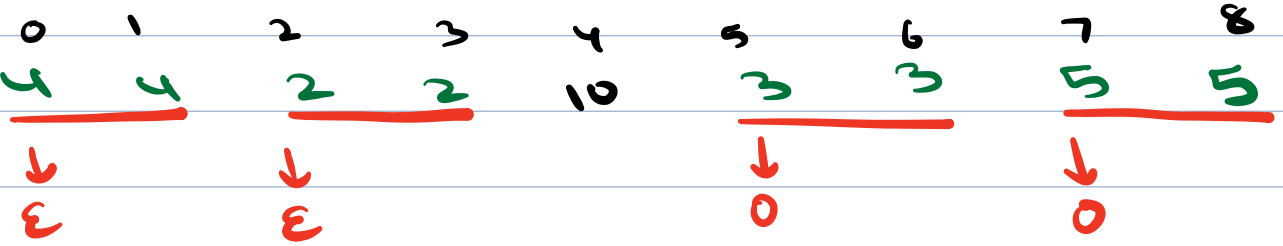
return A[mid]

② A[mid] is not unique



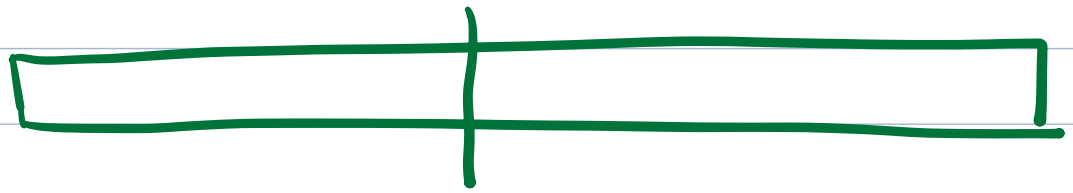


First occurrence of an elem in a pair → even idx



Before unique elem, every duplicate's first occurrence → even

After unique elem, every duplicate's first occurrence → odd



① A[mid] is unique element

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

② if (A[mid] == A[mid-1])
first = mid - 1

else
first = mid

if (first % 2 == 0) // Even

// before unique elem
go right, l = mid+1

else // odd

// after unique elem
go left, r = mid-1

ans = 19

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

l	r	mid	isAns	firstOcc	where to go
0	14	7	✗	6 → Even	Right l = mid+1
8	14	11	✗	11 → odd	Left r = mid-1
8	10	9	✗	9 → odd	Left r = mid-1
8	8	8	✓	return	A[8] ↓ 19

```
int findUnique(int a[], int N){
    int l=0, r=N-1
    while(l<=r){
        mid = (l+r)/2

        if((mid==0 || A[mid]!=A[mid-1]) &&
        (mid==N-1 || A[mid]!=A[mid+1]))
                return A[mid]

        // Mid ele is a duplicate
        firstOccurence = mid
        if(mid!=0 && A[mid]==A[mid-1])
                firstOccurence = mid-1

        if(firstOccurence%2==0){
                // Right
                l=mid+1
        }
        else{
                // Left
                r=mid-1
        }
    }
    return -1
}
```

TC: $O(\log_2 N)$

SC: $O(1)$

ans = -1

0	1	2	3	4	5
4	4	2	2	8	8

l	r	mid	isAns !
0	5	2	x

5. Given an array of N distinct elements, find any local minima in the array.

Local Minima: A no. which is smaller than its adjacent neighbors.

A : 3 6 1 0 9 15 8

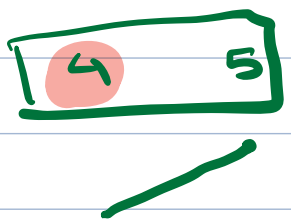
B : 21 20 19 17 15 9 7

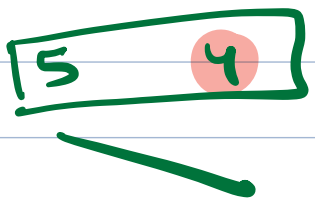
C : 5 9 15 16 20 21

D : 3 0 8

E : 5 8 12 3

N > 1





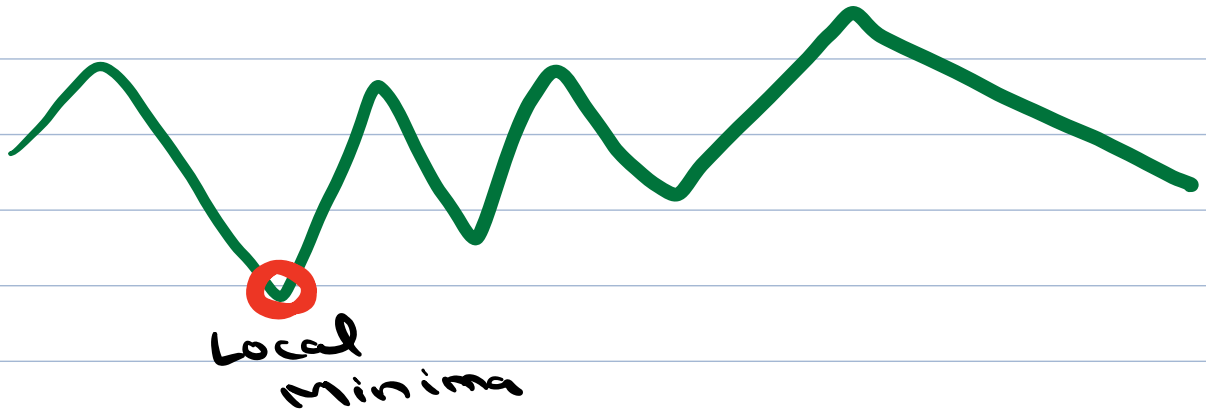
Local Minima always exist

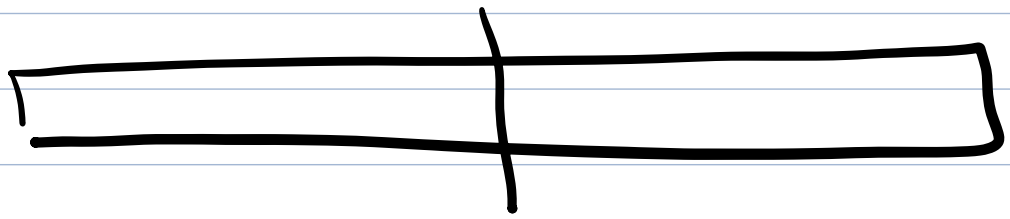
N = 1

Clarified from the interviewer/input Test cases

Target : Any local Minima

Search space : Array (0 → N-1 indx)

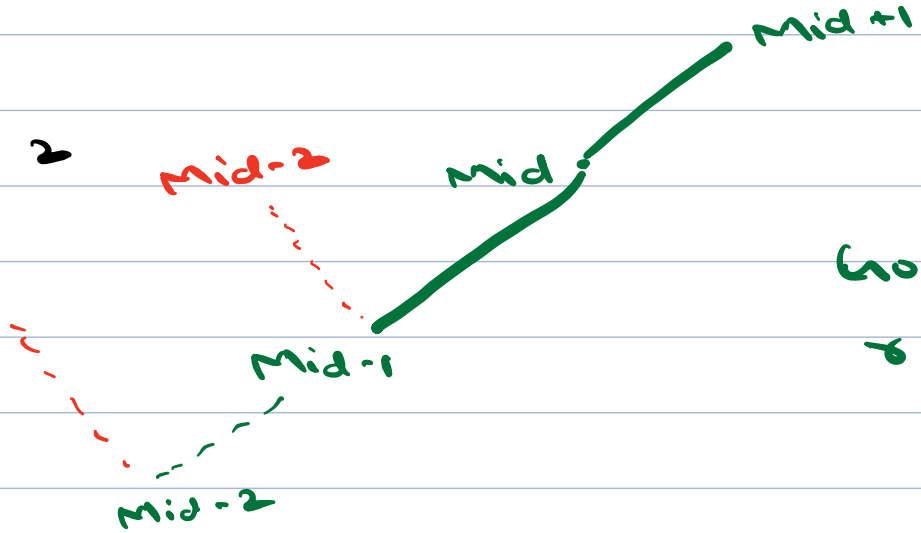




Case 1

return A[mid]

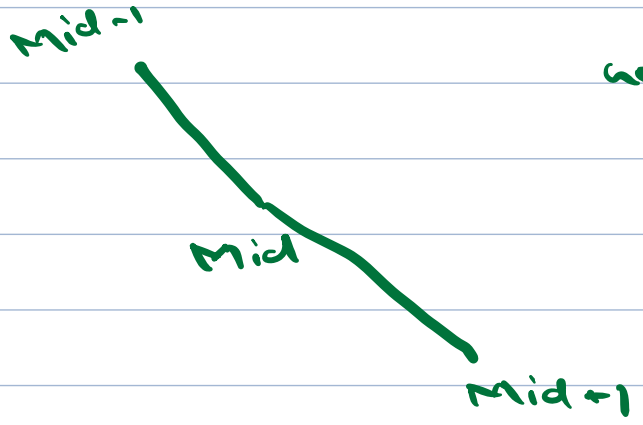
Case 2



Go left

h = mid-1

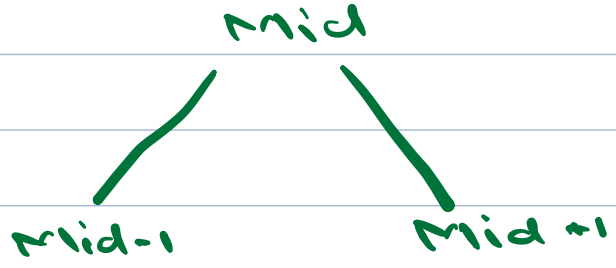
Case 3



Go Right

l = mid+1

Case 4



Go in any direction

```
int findLocalMinima(int a[], int N){

    l = 0, r = N-1
    while (l <= r) {
        mid = (l + r) / 2

Case 1  if(((mid==0) || A[mid] < A[mid-1])
          &&
         ((mid==N-1) || A[mid] < A[mid+1])))
            return A[mid]

Case 2 and 3  if(mid!=0 && A[mid] > A[mid-1]){
                  // Left
```

$r = mid - 1$

else

Case 3

$l = mid + 1$ // right

TC : $O(\log_2 N)$

SC : $O(1)$