Todays Content:

a) Searching Basics

b) Why mid at half

Problems

a) Search in sorted arr[]

b) Finding Floor in sorted arr[]

c) Finding 1st ocurence in sorted arr[]

d) Finding local maxima

Searching Basics

Bro/Sis missing → PoliceStation
- → Picture/.. ? What to search : Target
- → last seen .. ? Where to search : SearchSpace location

Example:

Target	SearchSpace
Word	{Dict / Books / News Paper}
PhoneNo	{Contact / Phonebook}

Obs: If SearchSpace is Ordered, Searching becomes easier

Search Dog in Dictionary

[A B [C D E F G] H I J K L] M N O ... X Y Z]

B < D, D : Dog, H > D, M > D

How to where to land? : land at mid

Why mid?

N | N/2 | N/2 : We always discard N/2

N | N/3 | 2N/3 : Best Discard 2N/3, Worst Discard N/3

Comparing worst case landing at mid better

When to apply BS:

After dividing search space into 2 parts, if we can discard 1 half of searchspace using some conditions, than we can apply binary search

N | N/2 | N/2 → N | N/2 | N/2

Note: N/2 | N/2 If we cannot discard, we cannot apply BS

Q) Given a sorted ar[N] search if k is present or not?

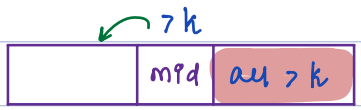
	0	1	2	3	4	5	6	7	8	9
ar[10] = {	3	6	9	12	14	19	20	23	25	27 }

k = 12 : Present

Idea1: Iterate on arr[] & check for k TC: O(N) SC: O(1)

Idea2: Target = Searching k Search Space = In arr[]

if ar[mid] == k : return true



if ar[mid] > k : discard right & goto left

if ar[mid] < k : discard left & goto right

> **Note:** We can use **space** to indicate searchspace

Eg1: ar[10] =

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

k = 12

l ↑ h ↑

l	h	m : (l+h)/2		
0	9	m = (0+9)/2 = 4	ar[m] > k :	goto left space h = m-1
0	3	m = (0+3)/2 = 1	ar[m] < k :	goto right space l = m+1
2	3	m = (2+3)/2 = 2	ar[m] < k :	goto right space l = m+1
3	3	m = (3+3)/2 = 3	ar[m] == k :	return true :

	0	1	2	3	4	5	6	7	8	9	
Eg1: ar[10] =	3	6	9	12	14	19	20	23	25	27	k = 21
							↑ h	↑ l			

l	h	m : (l+h)/2		
0	9	m = (0+9)/2 = 4	ar[m] < k :	goto right l = m+1
5	9	m = (5+9)/2 = 7	ar[m] > k :	goto left h = m-1
5	6	m = (5+6)/2 = 5	ar[m] < k :	goto right l = m+1
6	6	m = (6+6)/2 = 6	ar[m] < k :	goto right l = m+1
7	6	// Search space is done, break loop & return false		

```
bool search(int ar[], int k){ TC: O(log N/2) SC: O(1)
    int N = ar.length;
    int l = 0, h = N-1;
    while( l <= h ){
        int m = (l+h)/2
        if(ar[m] == k){ return true }
        if (ar[m] < k){ //discard left & goto right
            l = m+1
        else{ // discard right & goto left
            h = m-1;
    return false;
}
```

TC: Initially ~~~~~~~~~~ : n ele

After 1 iter : ~~~~~ : n/2 ele

After 2 iter : ~~ : n/4 ele

After 3 iter : _ : n/8 ele

⋮

After k iter : _ : 1 ele , k = $\log_2^N$ iterations ⇒ TC

29) Given a sorted arr[], find floor of a given num k

Floor : greatest ele <= k in arr[]

	0	1	2	3	4	5	6	7	8
arr[9] = {	-5	2	3	6	9	10	11	14	18 }
					*	*	*	*	*

k

5 : 3

4 : 3

10 : 10

-7 : Nothing

INT_MIN

24 : 18

Idea : Iterate on arr[] & calculate floor of k?

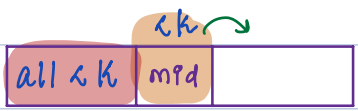
k = 8, ans = INT_MIN TC: O(N) SC: O(1)

i	arr[i]	ans
0	-5	-5 : better ans
1	2	2 : better ans
2	3	3 : better ans
3	6	6 : better ans
4	9	9 > 8 : cannot stop : return ans : 6

Idea2 : Target = Greatest ele <= k Search Space = arr[]

ans = INT_MIN

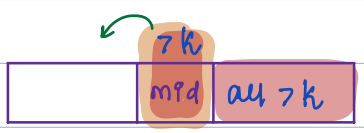
if arr[mid] == k : return k;



if arr[mid] < k :

ans = arr[mid]

l = mid + 1;



if arr[mid] > k :

h = mid - 1

	0	1	2	3	4	5	6	7	8	
Eg1: ar[10] =	-5	2	3	6	9	10	11	14	18	k=5
				l	h					

l	h	m : (l+h)/2	ans: -∞
0	8	m: (0+8)/2 = 4,	ar[m] > k: goto left h = m-1;
0	3	m: (0+3)/2 = 1,	ar[m] < k: ans = 2 goto right l = m+1
2	3	m: (2+3)/2 = 2	ar[m] < k: ans = 3 goto right l = m+1
3	3	m: (3+3)/2 = 3	ar[m] > k: goto left h = m-1;
3	2	: Search Space done, break : return ans = 3	

```
int floor(int ar[], int k){   TC: O(log N)  SC: O(1)
    int N = ar.length;
    int l=0, h = N-1, ans = INT_MIN
    while( l <= h){
        int m = (l+h)/2
        if(ar[m] == k){return k}
        if(ar[m] < k){
            ans = ar[m]; l = m+1
        }
        else //ar[m] > k
    }   }   h = m-1;
    return ans;
}
```

10:40pm

38) Given an sorted arr[], find the first occurence index of given element

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
arr[ ] = {	-5	-5	-3	0	0	1	1	5	5	5	5	5	5	8	10	10	15 }

ans = 10

k:

5 : 7

-5 : 0

20 : nothing : -1

Idea1: Iterate on arr[] get 1st occurence index

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

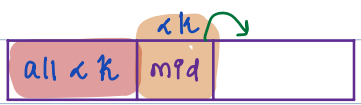
ele: 5 : 1st occurence index = 7 : logN
5 : last occurence index = 12 : logN
} frequency of single ele in sorted arr[]

Idea2:

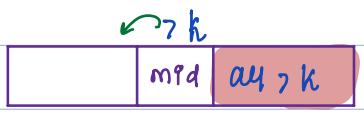
5 are [7..12] = 12 - 7 + 1 = 6 (a = 7, b = 12) TC: logN SC: O(1)

Target = 1st occurence index : k Search Space = arr[] ans = -1

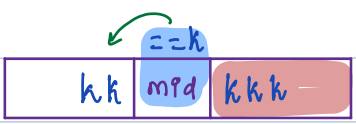
1st Occurence



if ar[mid] < k = go to right



if ar[mid] > k = go to left

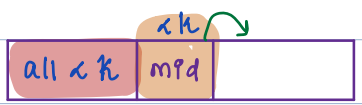


if ar[mid] == k =

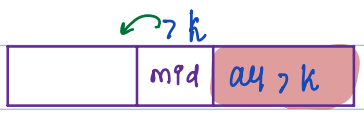
ans = mid;

go to left;

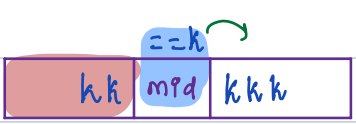
last Occurence



if ar[mid] < k = go to right



if ar[mid] > k = go to left



if ar[mid] == k =

ans = mid;

go to right

```
int firstOcc(int ar[], int k){ TC: O(logN) SC: O(1)
    int N=ar.length;
    int l=0, h=N-1, ans=-1;
    while(l<=h){
        int m=(l+h)/2
        if(ar[m]<k){l=m+1}
        else if(ar[m]>k){ h=m-1}
        else { //ar[m]==k,
            ans=m;
            h=m-1;
        }
    }
    return ans;
}
```

TODO:

1. Given sorted arr[] find last occr of k
2. Given sorted arr[] get freq of k

m-2	m-1	m	m+1	m+2
k	k	k		

Google:

Given a **unsorted arr[]** with all **distinct elements** return **any one** local maxima

localmaxima: An element is said to be local maxima, if > than it's adjacent elements {immediate left & right}

{ arr[i] > arr[i-1] && arr[i] > arr[i+1] }

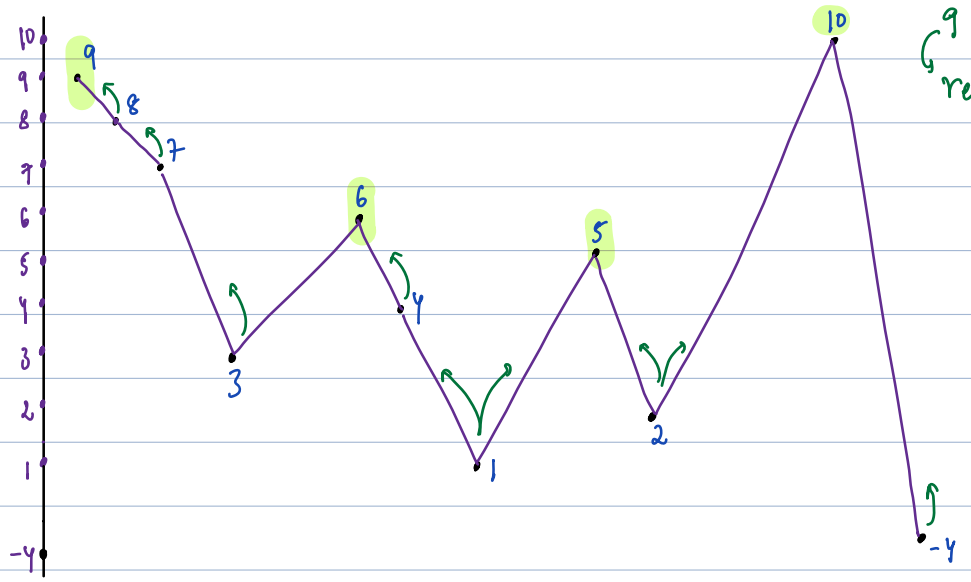
arr[0] > no left && arr[0] > arr[1]

arr[n-1] > arr[N-2] && arr[n-1] > no right

	0	1	2	3	4	5	6	7	8	9	10
Eg: arr[11] = {	9	8	7	3	6	4	1	5	2	10	-4 }

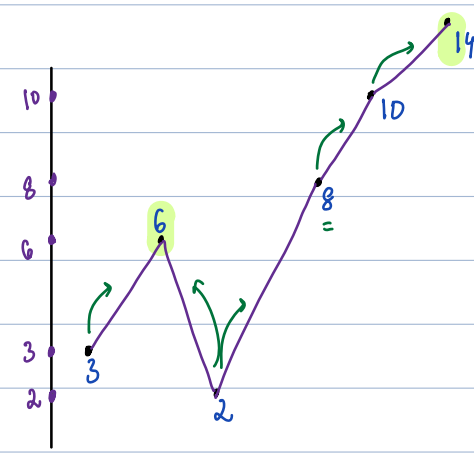
localmaxima: 9 6 5 10

return any one of them.



Eg: arr[6] = {3 6 2 8 10 14} localmaxima: 6 14

return any one



Q: What to return, if no local maxima?

{ Max of arr[] itself is a local maxima, We will atleast have 1 localmaxima }

Ideal: Iterate on arr[] & get max

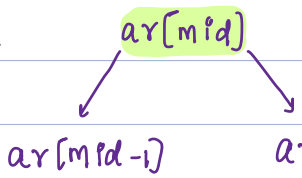
TC: O(N) SC: O(1)

Idea:

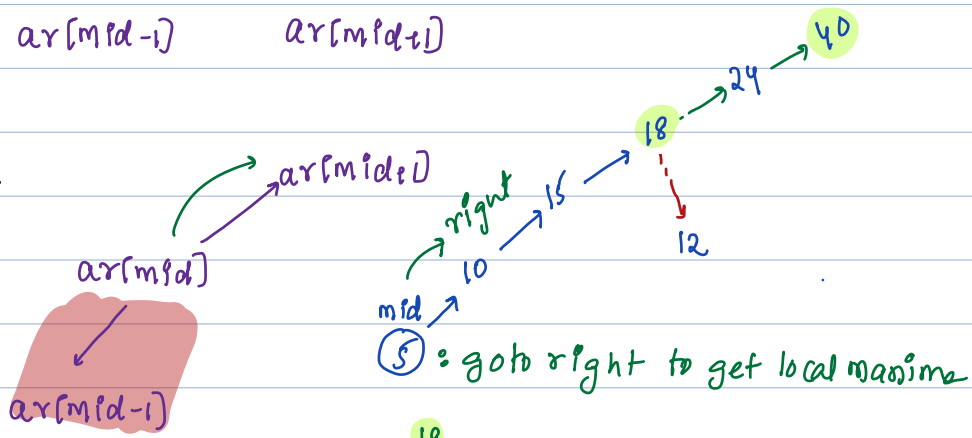
Target = local maxima Search Space = ar[]

Discard:

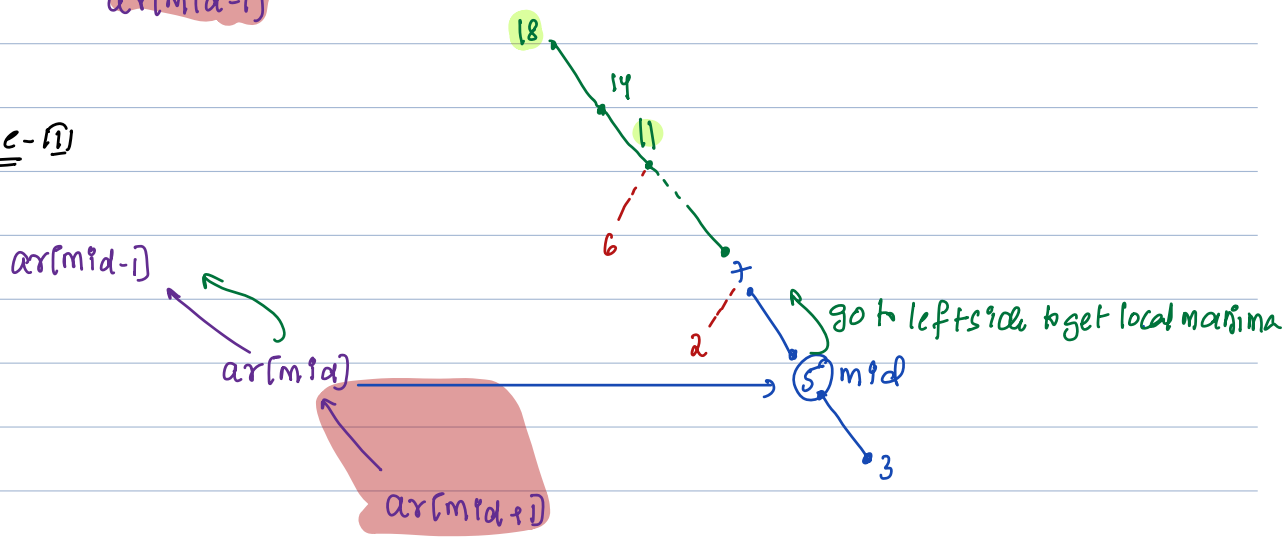
Case: I



Case-II



Case-III



obs: Goto side which increases, it will give us local maxima

```
int localmaxima(int ar[]){ TC: O(logN) SC: O(1)
    int N = ar.length;
    if(N==1) {return ar[0]}
                          // if N==1 : can cause error
    if(ar[0] > ar[1]) {return ar[0]}
    if(ar[N-1] > ar[N-2]) {return ar[N-1]}
    int l=1 h= N-2; // change in l & h
    while(l<=h){
        int m = (l+h)/2
        if(ar[m] > ar[m-1] && ar[m] > ar[m+1]){
            return ar[m]
        }
        if( ar[m] < ar[m-1]){
            h = m-1; goto left
        }
        else{
            l = m+1; goto right
        }
    }
}
```

ar[m] > ar[m-1] : m = 0 ⇒ ar[0] > ar[-1] ❌

ar[m] > ar[m+1] : m = n-1 ⇒ ar[n-1] > ar[N] ❌