Todays Content:

- Searching Basics
- Why mid at half ?

Problems:

a) Search in sorted arr[]

b) finding floor in a sorted arr[]

c) finding 1st occurence in sorted arr[]

d) finding local minima

Searching Story :

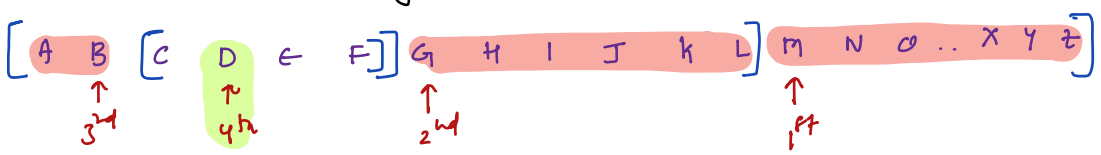
Bro/Sis → Police → photo / details (What to search) : Target
Police → last location (Wher to search) : Search Space

Example :

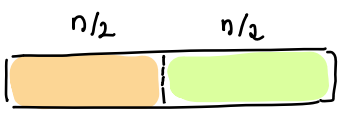
Target	Search Space
Word ⟶	{ Dict / Books / News paper }
phone number ⟶	{ Contacts / phone books }

Obs : If Search Space is ordered, Searching becomes easier

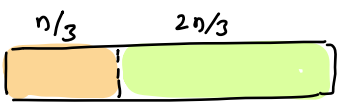
Search Dog in Dictionary :



Why land at mid ?



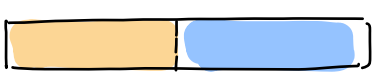
: Always we can discard N/2 : It is better



: Worst Case we can only discard N/3

When to apply BS :

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



:

Note: If we cannot discard one half of search space, we cannot apply BS

1(a) 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

Idea1: linear search on ar[] TC: O(N) SC: O(1)

Idea2:

Case-I:

ar[mid]==k: return True

Case-II:

ar[mid] > k:
discard right
search left

Case-III:

ar[mid] < k:
discard left
goto right

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

m, m, l h, m

l	h	m = (l+h)/2	
0	9	4	ar[4] > 12 : goto left : h = m-1
0	3	1	ar[1] < 12 : goto right : l = m+1
2	3	2	ar[2] < 12 : goto right = l = m+1
3	3	3	ar[3] == 12 : return True

ar[10] = { 0:3, 1:6, 2:9, 3:12, 4:14, 5:19, 6:20, 7:23, 8:25, 9:27 } k = 21

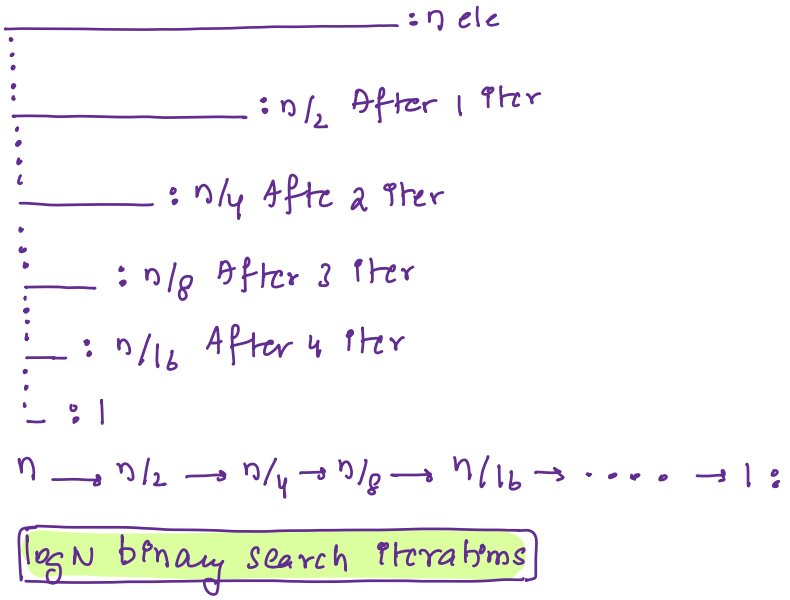
m, m, l h, m

l	h	m = (l+h)/2	
0	9	4	: ar[4] < 21 : goto right : l = m+1
5	9	7	: ar[7] > 21 : goto left : h = m-1
5	6	5	: ar[5] < 21 : goto right : l = m+1
6	6	6	: ar[6] < 21 : goto right : l = m+1
7	6	:	l > h : break, return false

↳ our search space finished

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

TC:



$n \to n/2 \to n/4 \to n/8 \to n/16 \to \cdots \to 1$:

logN binary search iterations

2Q) Given a sorted arr[] find floor of given num k

greatest ele $\leq k$ 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

k = 4 : 3

k = 10 : 10

k = -7 : nothing → INT_MIN

refer 1 click

k = 24 : 18

k = 8 ans = INT_MIN

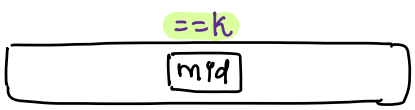
i :	arr[i] :	ans
0	-5 :	-5
1	2 :	2
2	3 :	3
3	6 :	6 : return 6
4	9 > 8 :	9 cannot be ans, break because
5	10 > 8	everything in right also cannot be ans

Idea: Iterate in arr[], check if arr[i] can be ans & update accordingly

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

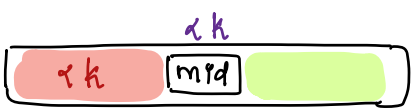
Idea2:

Case-I



arr[mid] == k return k

Case-II

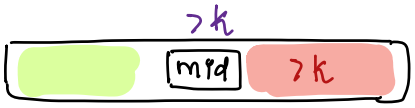


arr[mid] < k :

ans = arr[mid]

goto right

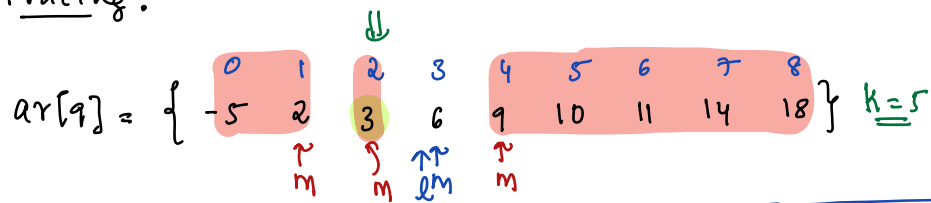
Case-III



arr[mid] > k :

goto left

Tracing:



ans = INT_MIN

l	h	m	
0	8	4	ar[4] > k : goto left, h = m-1
0	3	1	ar[1] < k : goto right, l = m+1 ans = ar[1] = 2
2	3	2	ar[2] < k : goto right, l = m+1 ans = ar[2] = 3
3	3	3	ar[3] > k : goto left, h = m-1
3	2	: l > h break : return ans = 3	

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

TODO: ceil(k): return Smallest number >= k in arr[]

30) Given a **sorted ar[]** find first occurence index of **given element**?

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
ar[] = {	-5	-5	-3	0	0	1	1	5	5	5	5	5	5	5	8	10	10	15	15 }

k : 1st occ

5 : 7

-5 : 0

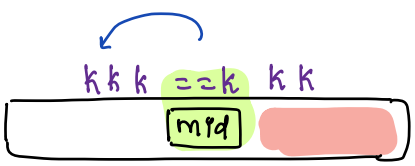
20 : nothing
return -1

Idea: Iterate on ar[], check first index with k

↳ TC: O(N) SC: O(1)

Idea2:

Case 1



ar[mid] == k

ans = mid

goto leftside

Case2

ar[mid] > k :

goto leftside

Case3

ar[mid] < k :

goto rightside

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

k = 5

l h m

ans = -1 TODO

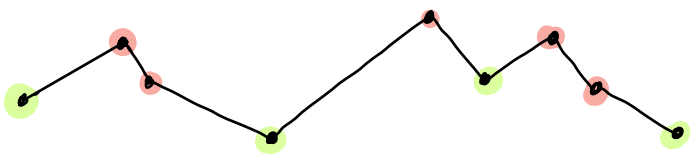
```
int firstOcc(int ar[N], int k){   TC: O(log N)  SC: O(1)
    l = 0, h = N-1, ans = -1
    while(l <= h){
        m = (l+h)/2
        if(ar[m] == k){
            // update & goto left
            ans = m // updating with index, because
            h = m-1   that's our target
        }
        else if(ar[m] > k){
            //goto left
            h = m-1
        }
        else{ // ar[m] < k
            // goto right
            l = m+1
        }
    }
    return ans;
}
```

TODO: find last occurence index of k

10:45 → 10:55 break

40) Given unsorted ar[N] distinct elements (N>=2) return any local minima

An element is said to be local minima, if its less than < than it's adjacent elements



Adjacent nodes

$$\begin{bmatrix} ar[i-1] > ar[i] < ar[i+1] \\ ar[0] < ar[1] \\ ar[n-2] > ar[n-1] \end{bmatrix}$$

Eg: ar[8] = { 9, 8, 7, 3, 6, 4, 1, 5, 2 } (indices 0 1 2 3 4 5 6 7 8)

return any one { 3, 1 2 }

Ideal: Iterate on arr[], check if a element ar[i] is local minima or not, but comparing with adj elements

→ TC: O(N) SC: O(1)

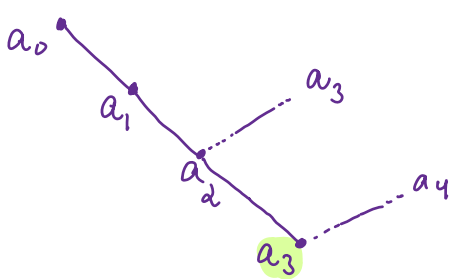
Will we always have local minima ? We will have atleast 1 local minima

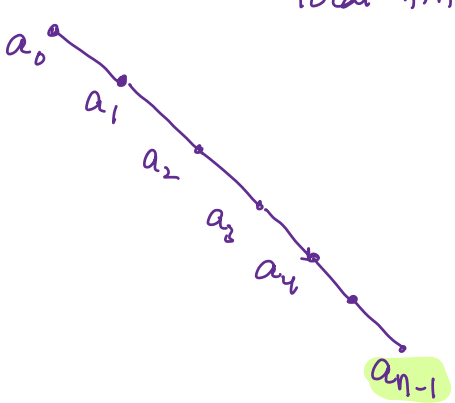
Case-1:

Case-2:

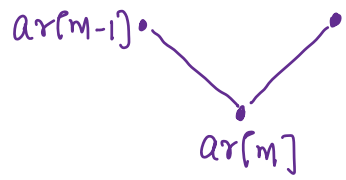
Obs1: Say if data increases even at one index pos that itself is local minima

Obs2: If data continously decreas, last element is local minima



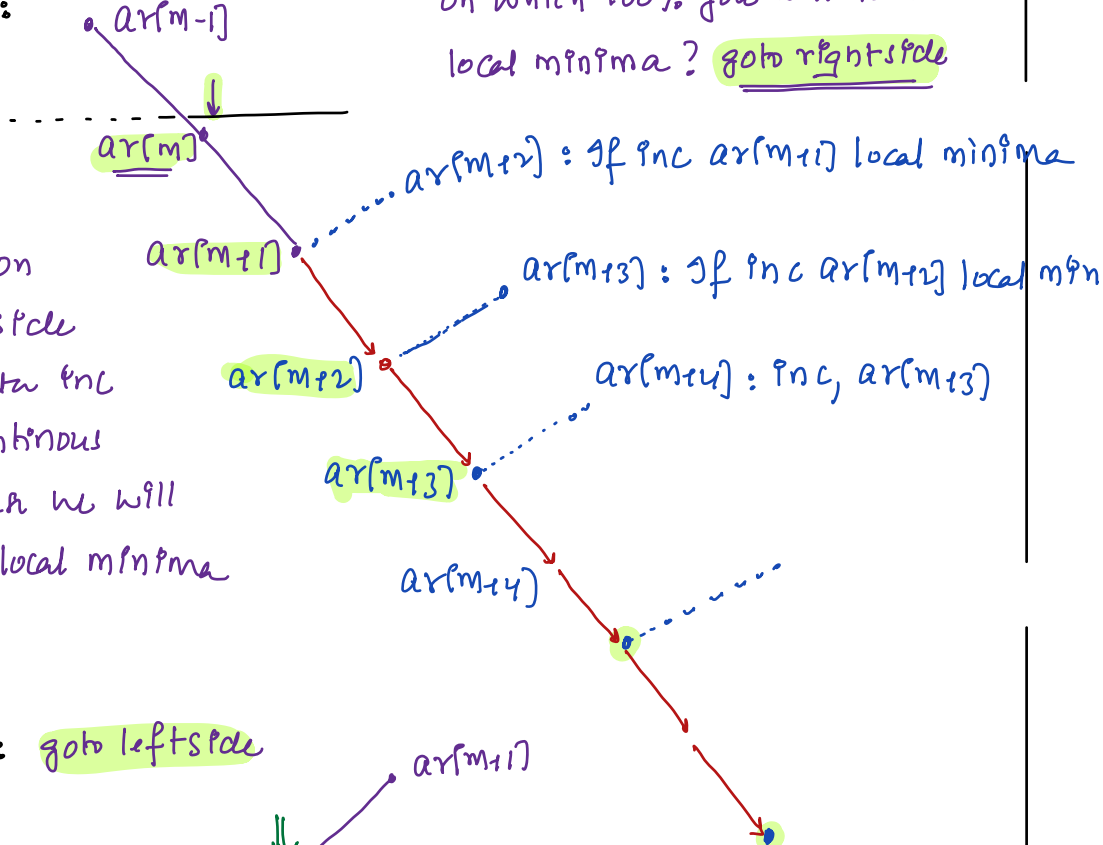


Case-I : ar[m-1] ar[m+1] ar[m] itself is local minima



ar[m-1] > ar[m] &&
ar[m] < ar[m+1]

Case-II : on which 100% you will have local minima ? goto rightside



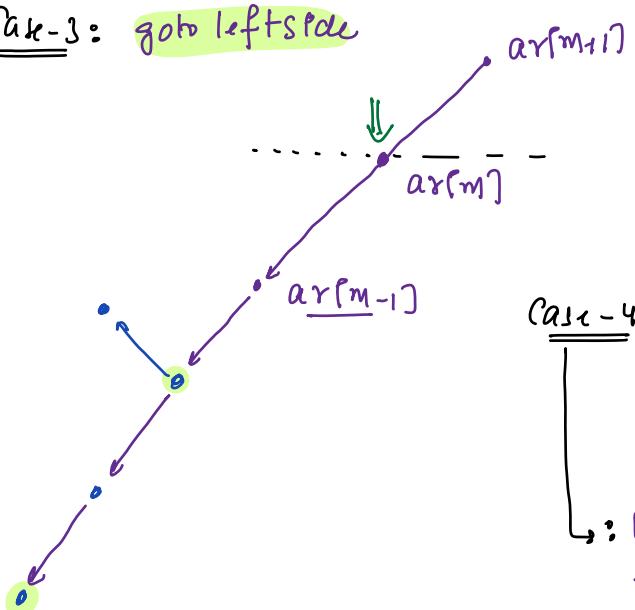
ar[m+2] : If inc ar[m+1] local minima

ar[m+3] : If inc ar[m+2] local min

ar[m+4] : inc, ar[m+3]

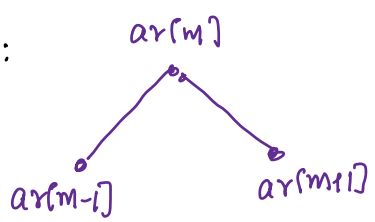
Obs : on rightside If data inc & continous decrease we will have local minima

Case-3 : goto leftside



Final : Which ever side data is decreasing we goto that side

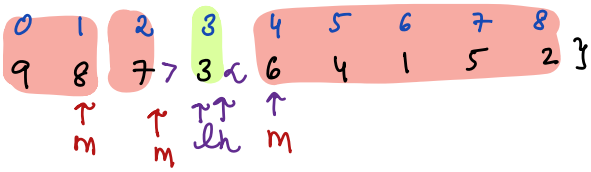
Case-4 :



: We can go any side because there is local minima in both sides

: goto leftside

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

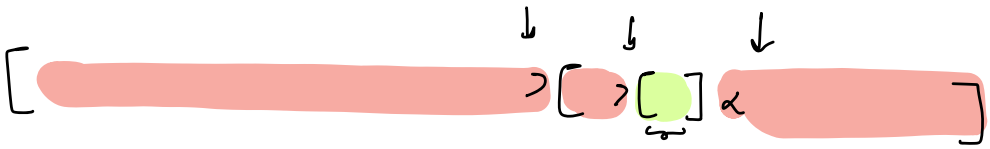


l	h	m	:	data dec
0	8	4	: ar[m-1] < ar[m] : left : goto left : h = m-1	
0	3	1	: ar[m] > ar[m+1] : right : goto right = l = m+1	
2	3	2	: ar[m] > ar[m+1] : right : goto right = l = m+1	
3	3	3	: ar[m] is local minima return ar[m]	

```
int localMin(int ar[N]){   TC: O(log N)  SC: O(1)
                ↳ N >= 2
    if (ar[0] < ar[1]) { return ar[0] }      // Edge Cases
    if (ar[n-1] < ar[n-2]) { return ar[n-1] }

    l = 1, h = N-2  // update to avoid edge cases

    while (l <= h){
        m = (l+h)/2

        if (ar[m-1] > ar[m] && ar[m] < ar[m+1]){
            return ar[m]
        }
        if (ar[m-1] < ar[m]){          // going to side which
            // goto left                  decrease, because
            h = m-1                       we are 100% it
        }                                 will contain local
        else // ar[m] > ar[m+1]           minima
            // goto right
            l = m+1
        }
    }
}
```

Edge Cases:
: m == 0 : ar[-1] > ar[0] : error
: m == n-1 : ar[n-1] < ar[n] : error

→ malls :



→ 2^{31}

$m = \frac{l+h}{2}$ or $m = l + \frac{h-l}{2}$

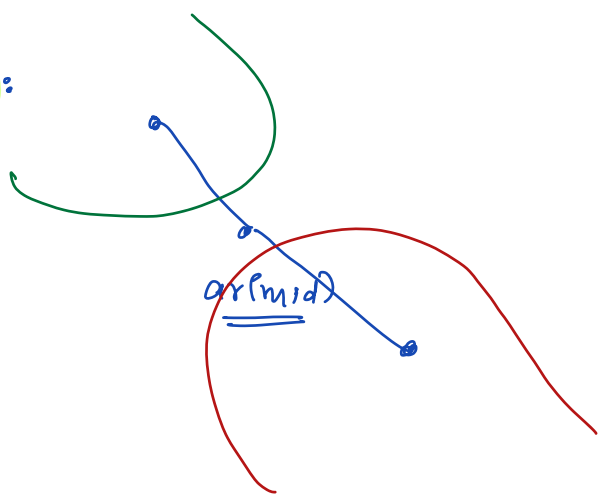
l h

$m = \frac{l}{2} + \frac{h}{2}$ $l=1$ $h=5$ → $m=4$

$\frac{l}{2} + \frac{h}{2} \rightarrow \frac{1}{2} + \frac{5}{2} \rightarrow 3$

→ any minima

→ 1st minima:



During prep

→ Topic:

→ Leetcode

→ Easy / medium / Above med / Hard : 45min Sha stop 4 thinks / you're in / understand

Before Interv:

→ previous asked questions

→ review notes