Todays Content:

Oct 4th holiday: Navami ✓

- 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 : Whom to Search : Target
- last location : Where to Search : Search Space

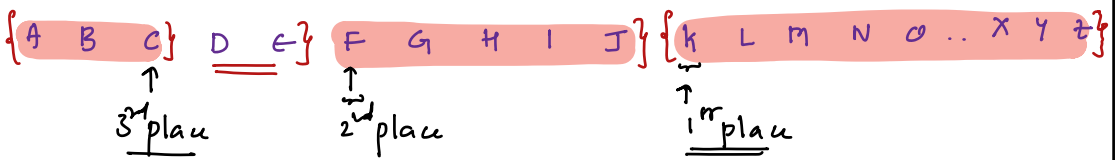
Example :

Word ⟶ { Dict / Books / News paper }

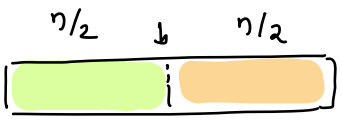
phone number ⟶ { Contacts / phone books }

If search space is ordered / searching become easier

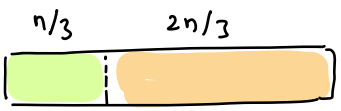
Search Dog in Dictionary :



Why land at mid ?



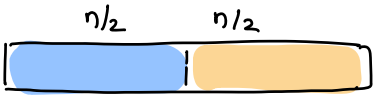
: atleast we discard 50%.



: atleast we discard 33%.

When to apply BS :

After dividing search space into 2 halves we should be able discard 1 halfs of search space using some conditions.



Note: if we are not able to discard we cannot apply BS

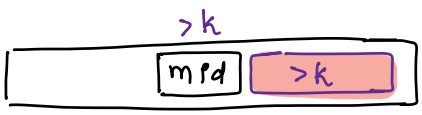
Q) Given a sorted ar[N] search if k is present or not?

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

Idea1: linear Search TC: O(N) SC: O(1)

Idea2: Check, if we discard a half

Case-I: ar[mid] == k return True



Case-II: ar[mid] > k:

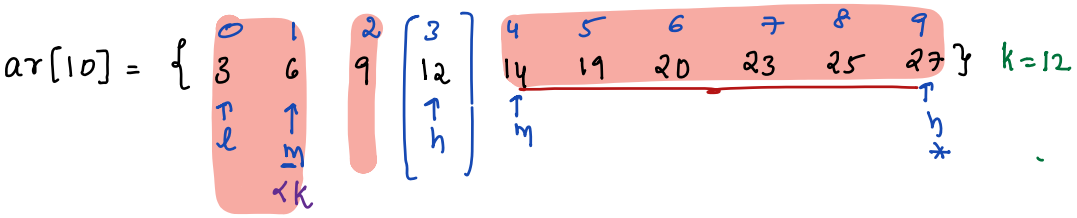
→ discard right

→ search left

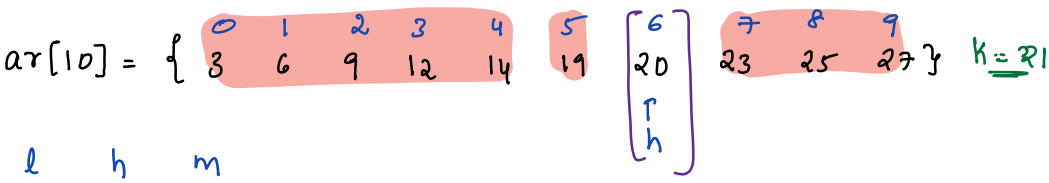
Case-III: ar[mid] < k

→ discard left

→ search right

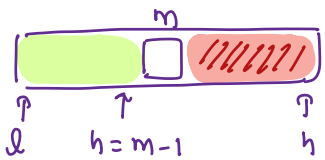


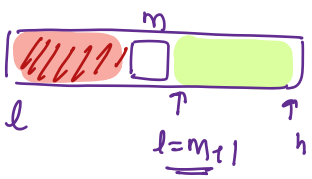
l	h	m = ⌊(l+h)/2⌋	k=12
0	9	4	→ ar[m] > k : search left, h = m-1
0	3	1	→ ar[m] < k : search right, l = m+1
2	3	2	→ ar[m] < k : search right, l = m+1
3	3	3	→ ar[m] == k : return True



l	h	m	
0	9	4	: ar[4] < k : search right = l = m+1
5	9	7	: ar[7] > k : search left = h = m-1
5	6	5	: ar[5] < k : search right = l = m+1
6	6	6	: ar[6] < k : search right = l = m+1
7	6		: l > h Stop return False

```
bool search(int ar[N], int k)
{
    l = 0, h = N-1
    while(l <= h) {
        m = (l+h)/2
        if(ar[m] == k) { return True }
        if(ar[m] > k) {
            // search left
            h = m-1
        }
        else {
            // search right
            l = m+1
        }
    }
    return false
}
```





TC:

N Elements

↓ After 1 iteration

N/2

↓ After 1 iteration

N/4

↓ After 1 iteration

N/8

⋮

→ 1

$N \rightarrow N/2 \rightarrow N/4 \rightarrow N/8 \rightarrow \dots 1$

$= \log_2 N$ 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
ar[9] = {	-5	2	3	6	9	10	11	14	18 }

k = 5 : 3

k = 4 : 3

k = 10 : 10

k = -7: nothing

↳ INT_MIN

k = 24 : 18

k = 12 ans = INT_MIN

i : ar[i]

0 -5 : ans = -5

1 2 : ans = 2

2 3 : ans = 3

3 6 : ans = 6

4 9 : ans = 9

5 10 : ans = 10

6 11 : ans = 11

7 14 : 14 cannot be ans, break return ans

Ideal: Iteration from left → right update ans

↳ TC: O(N)

Ideal:

Case-I: ar[mid] == k : return k

Case-II: ar[mid] < k : ans = ar[mid] goto right

Case-III ar[mid] > k : goto left

Tracing:

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

ans = INT_MIN

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

```
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 k;}
        if(ar[m] < k){
            ans = ar[m], goto right
            l = m+1
        }
        else{ // ar[m] > k
            goto left
            h = m-1
        }
    }
    return ans;
}
```

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

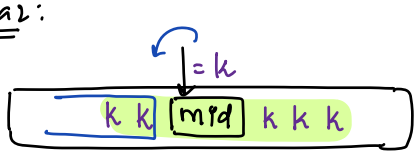
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
ar[ ] = {	-5	-5	-3	0	0	1	1	5	5	5	5	5	5	5	8	10	10	15	15 }

k :	1st occ
5	7
-5	0
20	-1 → no element

Idea: Iterate in array from [0, n-1] & get 1st occurence index of k

→ TC: O(N) SC: O(1)

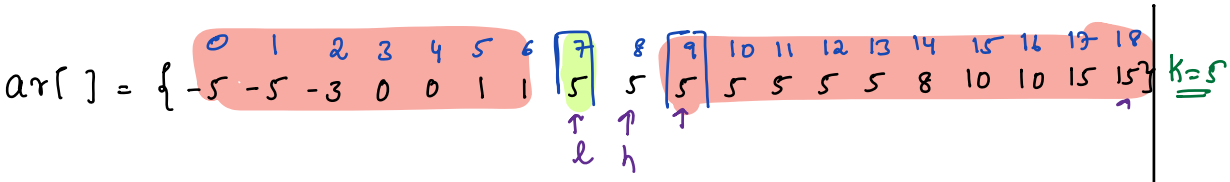
Idea2:



Case-I: ar[mid] == k : ans = mid, goto left

Case-II: ar[mid] < k : goto right

Case-III ar[mid] > k : goto left



ans = -1

l	h	m		
0	18	9	ar[m] == k, goto left h = m-1	ans = 9
0	8	4	ar[m] < k, goto right l = m+1	
5	8	6	ar[m] < k, goto right l = m+1	
7	8	7	ar[m] == k, goto left h = m-1	ans = 7
7	6	: break return ans		

```
int firstOccur(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){
            ans = mid, goto left
            h = m-1
        }
        else if(ar[m] < k){
            goto right
            l = m+1
        }
        else // ar[m] > k
            goto left
            h = m-1
    }
    return ans;
}
```

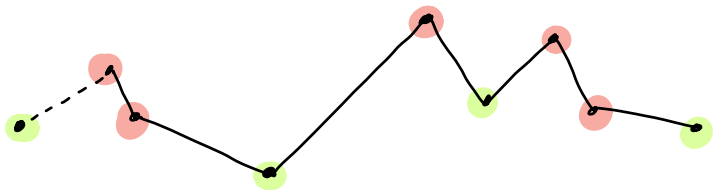
TODO: last occurence index of element = k

8:32 ⟶ 8:40 break

long

40) Given unsorted ar[N] distinct elements return any local minima

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



i-1	i	i+1
	0	1
n-2	n-1	

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

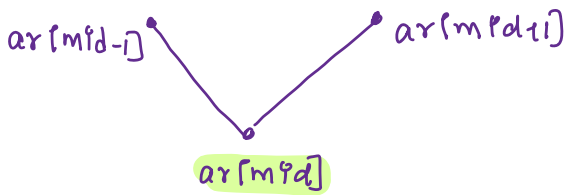
0	1	2	3	4	5	6	7	8
9	8	7	3	6	4	1	5	2
*	*	*	✓	*	*	✓	*	✓

return any local minima
ans = {3, 1, 2}

Idea1: Iterate on arr[], check if a particular element is local minima or not? TC: O(N) SC: O(1)

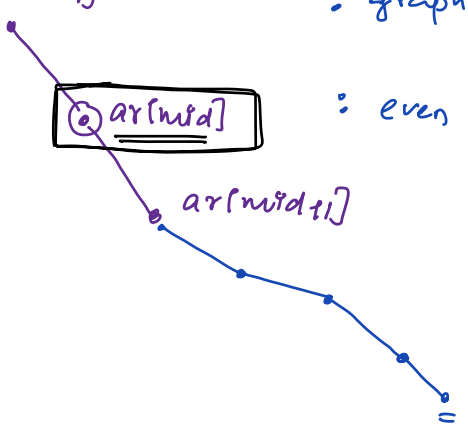
Idea2:

Cases-[:



Case-1: goto rightside: discard leftside

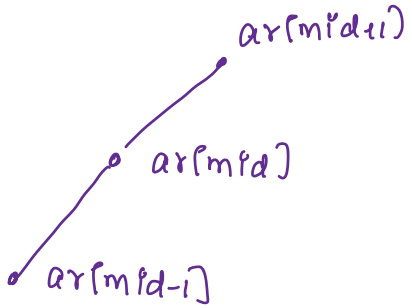
ar[mid-1]



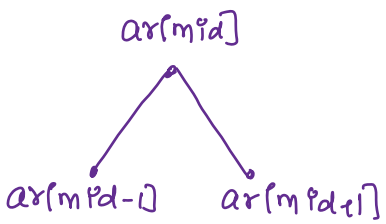
: graphs always decrease, last element is local minima

: even if there is an increase, that element is local minima

Case-[iii]: goto leftside, discard rightside



Case-[iv]: goto anyside we will find local minima



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

0	1	2	3	4	5	6	7	8
9	8	7	3	6	4	1	5	2
			↑ m					

l	h	m	
0	8	4	: if (ar[mid] > ar[mid-1]) goto left h = m-1
0	3	1	: if (ar[mid] > ar[mid+1]) goto right l = m+1
2	3	2	: if (ar[mid] > ar[mid+1]) goto right l = m+1
3	3	3	: ar[mid] is local minal return ar[mid] = 3

```
int localMin(int ar[N]){   T.C: O(log N)   S.C: O(1)
                  ↳ N>=2

TODO: local maxima

EdgeCase:
[ if(ar[0] < ar[1]) return ar[0]
  if (ar[n-2] > ar[n-1]) return ar[n-1] ]

l=1, h=n-2
while(l<=h){
    m=(l+h)/2
    if (ar[m-1] > ar[m] && ar[m] < ar[m+1]){   // if m==n+1 edge case
        return ar[m];
    }
    if( ar[m-1] < ar[m]){                      // if m==0 edge case
        //goto left
        h=m-1
    }
    else // ar[m] > ar[m+1]
        //goto right
        l=m+1
    }
}
```