Binary Search - 1

Searching → target: what to search?
Search space: where to search?

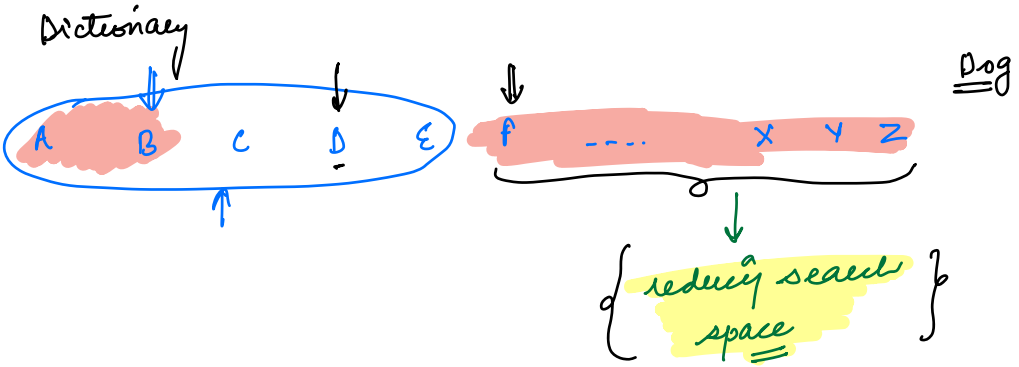
= word
phone no
Book

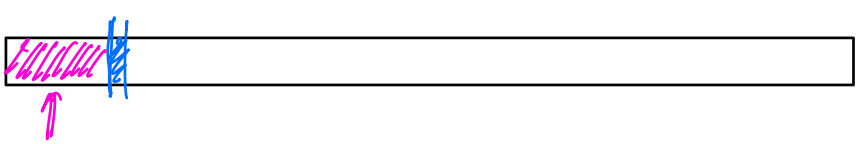
dictionary
list - by your friend's shelf

newspaper
phonebook dir
library

arrangement

newspaper → linear search - O(n)



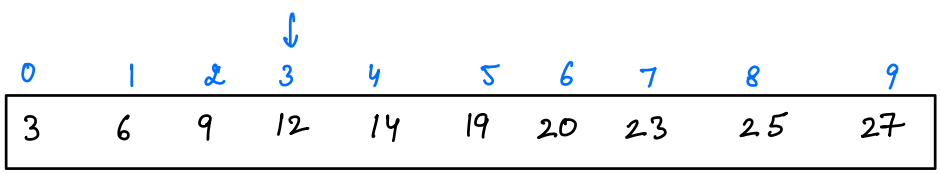


Binary search – breaking your search space into two parts

↓

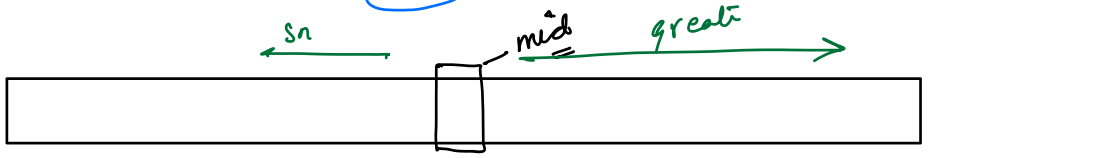
rejecting one of the parts on the basis of some condⁿ

Q given a sorted array. Find index of K. (If K doesn't exist, return -1)



K = 12 → 3

B.F: → linear search - O(n)

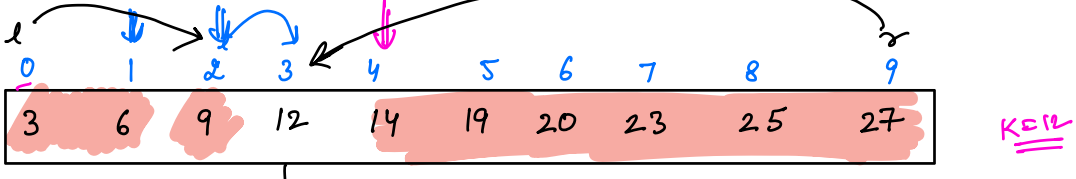


6 12

arr[mid] < target → reject left part → search in right

arr[mid] == target return mid;

arr[mid] > target → reject right → search in left



K = 12

search space →

l	r	mid
0	9	4
0	3	1
2	3	2
3	3	3

14 > 12 → ignore right, r = mid - 1

6 < 12 → l = mid + 1

9 < 12 → l = mid + 1

12 == 12 → target ✓

1) either you find your target
2) search space vanishes

```
int binarySearch(int arr[], int n, int target)
{
    // define your search space
    int l=0, r=n-1;

    while( l<=r )
    {
        int mid = (l+r)/2;
        if(arr[mid]==target)
            return mid;
        else if(arr[mid]<target)
        {
            l=mid+1;
        }
        else
            r=mid-1;
    }

    return -1;
}
```

T.C: $N \longrightarrow N/2 \longrightarrow N/4 \longrightarrow N/8 \longrightarrow \cdots \longrightarrow 1$

$\rightarrow \log N$

S.C: $O(1)$

$N = 10^9$, $\log 10^9 \approx 30$

Q given a sorted array. Find floor of given element k.

greatest element <= k

-5	2	3	6	9	10	11	14	18

-5 < -7

3 <= 12

K = 5 → 3
K = 4 → 3
K = 10 → 10
K = 20 → 18
K = -7 → INT_MIN

k = 12

ans = -5 2 3 6 9 10 10 11

B.F: linear search until you are getting better ans — O(N)

mid

target = k

arr[mid] < K
ans = arr[mid]
l = mid+1

arr[mid] == k
return k;

arr[mid] > k
r = mid-1

k = 12, 18

0	1	2	3	4	5	6	7	8
-5	2	3	6	9	10	11	14	18

K = 5

ans = INT_MIN

l	r	mid		
0	8	4	9 > 5	r = mid-1
0	3	1	2 < 5	ans = 2, l = mid+1
2	3	2	3 < 5	ans = 3, l = mid+1
3	3	3	6 > 5	r = mid-1
3	2			

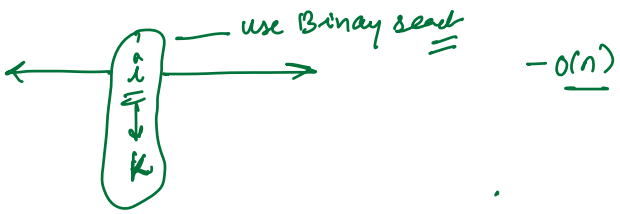
ceil → >= k

Q given a sorted array of N elements. Find the frequency of element K.

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

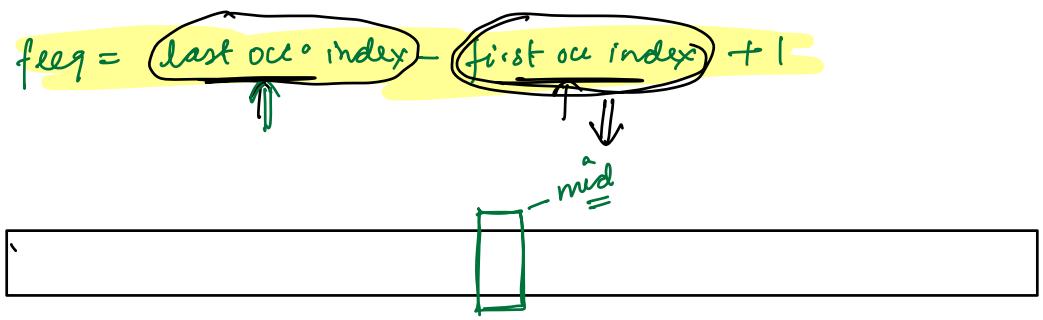
K=5

B.F: → linear search → O(n)



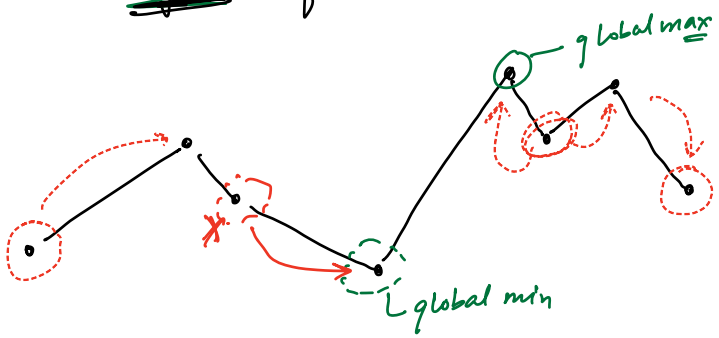
- O(n)

freq = last occ° index − first occ index + 1



T.C: log(n) + log n ⇒ O(log n)

Q given an unsorted array of N distinct elements.
Find any one of the local minima.



local minima
⇓
both neighbours have to be larger

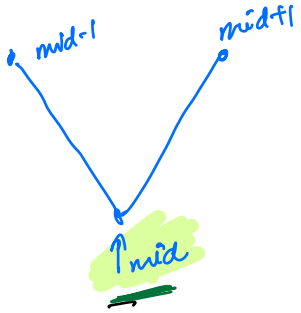
any no of local minima
local minima always exist

>=1

arr[i-1] > arr[i] < arr[i+1]

B.F :- linear search - O(N)

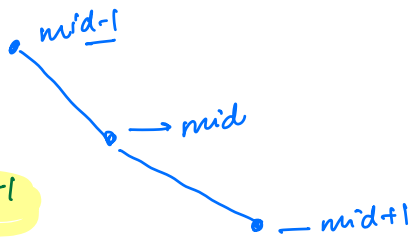
case I:



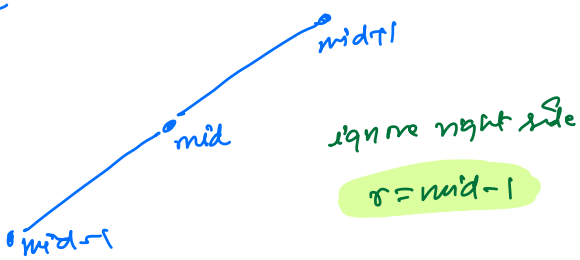
case II

ignore left

l = mid+1



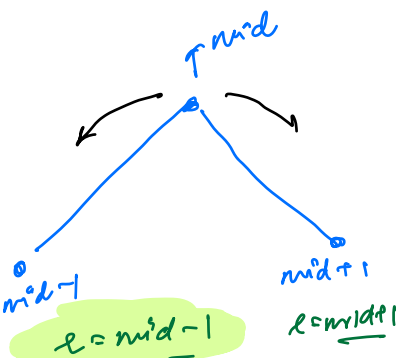
case III



ignore right side

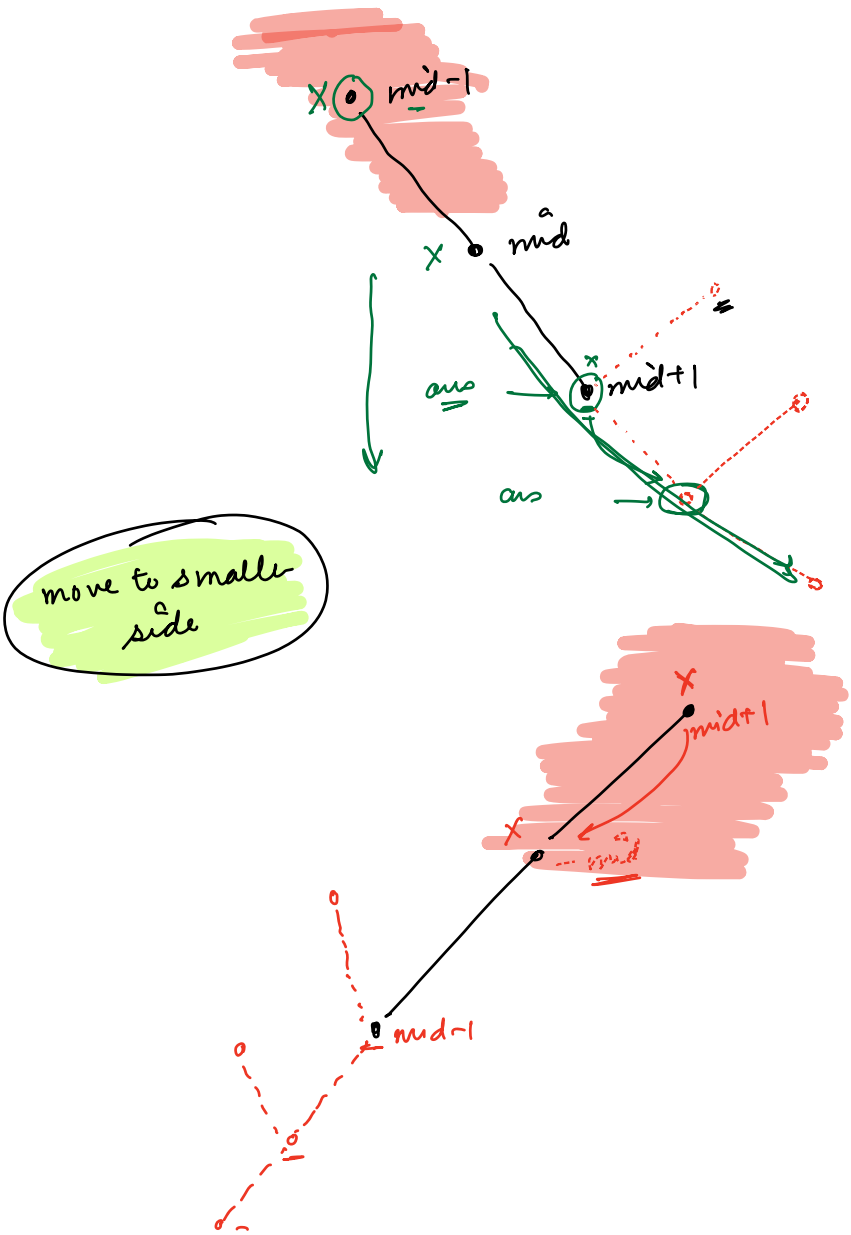
r = mid-1

case IV

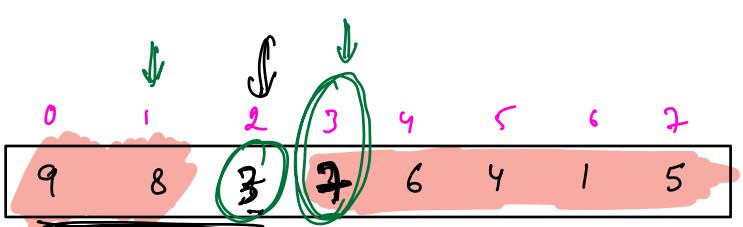


l = mid-1

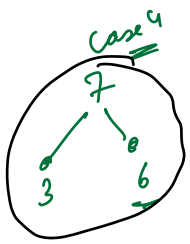
l = mid+1

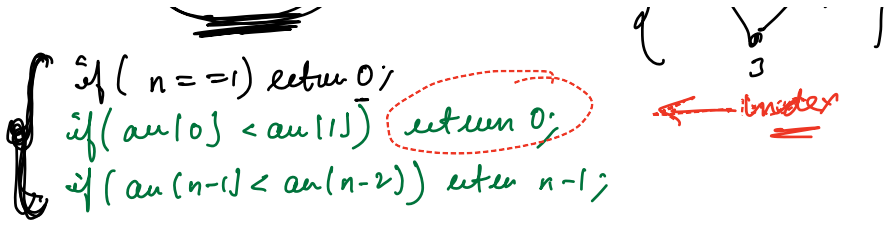


There has to be one local minima on right side



l	r	mid	
0	7	3	$l = mid-1$ $l = 3-1 = 2$
0	2	1	$l = mid+1$
2	2		→ ans





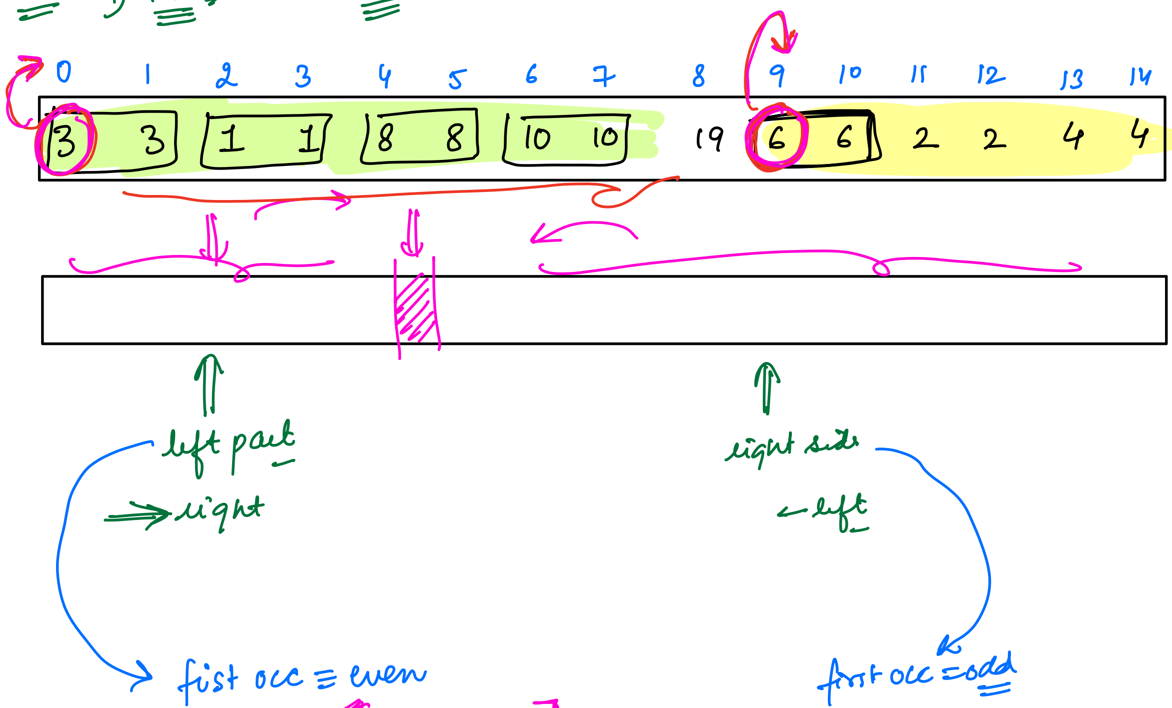

```
if ( n == 1) return 0;
if( arr[0] < arr[1]) return 0;
if ( arr[n-1] < arr[n-2]) return n-1;

int l=1, r= n-2;
  while ( l <= r)
  {
        int mid = (l+r)/2;
        if ( arr[mid-1] > arr[mid] && arr[mid+1] > arr[mid])
              return mid;
        else if ( arr[mid-1] > arr[mid])
           {        l= mid+1; }
        else      r= mid-1;
  }
```

Q Every element occurs twice except for one. Find unique element.

Note: Duplicates are adjacent to each other.

array is not sorted

B.F: 1) Take XOR. — O(n)



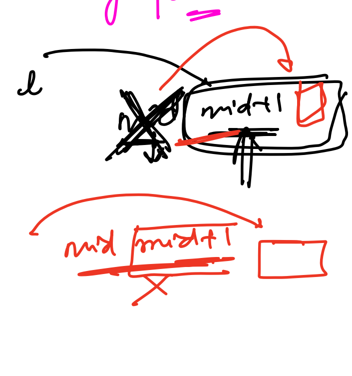
[Base cases]

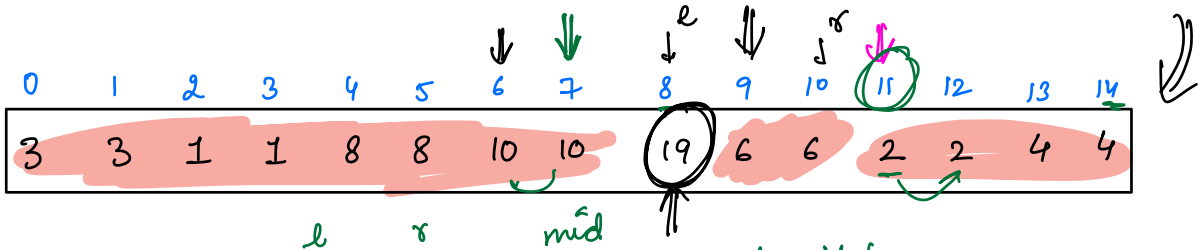
$X \oplus X$

fact
```
if ( arr[mid-1] != arr[mid] && arr[mid+1] != arr[mid])
    return arr[mid];
```

first occ
```
if( arr[mid-1] == arr[mid])  → mid is not my first
    mid = mid-1;
```

```
if ( mid % 2 == 0)
{     // left part
      l = mid+2;
}
else   r = mid-1;
```





l	r	mid	
0	14	6	mid = mid-1, left, towards l = mid+2
8	14	11	odd → l = mid-1
8	10	6	odd → l = mid-1

Double