Searching - 1

1) Target (what to search)
2) Search Space (where to search)

Target

T-shirt

Search Space

Room
Bed
wardrobe

Factors that affect the time taken to search?

1) size of search space
2) order of elements in search space ✓

Target

Word

Phone Number

Search Space

Newspaper Vs Dictionary

Hand written Vs Phone book.
List ordered

N words in Newspaper

Search in News Paper ?
Search word by word. ?

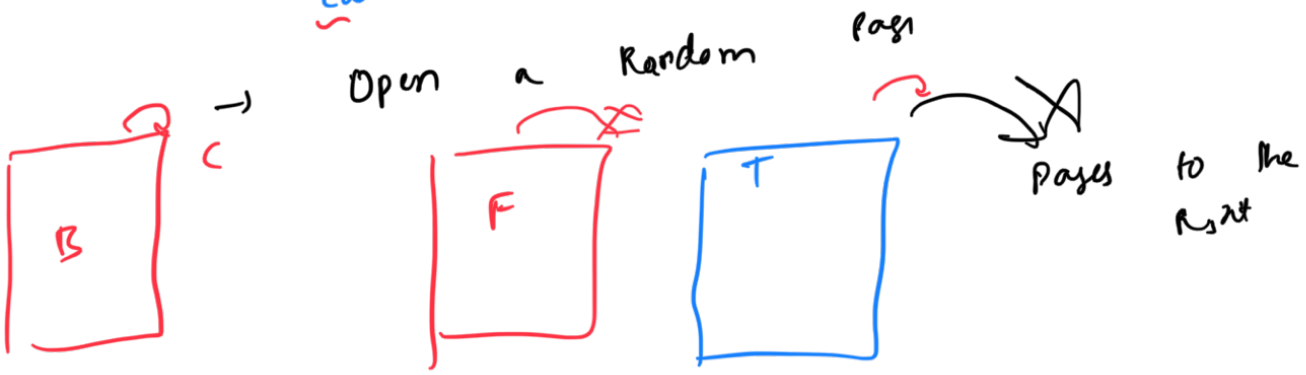
Linear Search : O(n)

How is ordered arrangement helping us to search faster ?

⇒ We are able discard some search space in every

→ Search for a word in

cat



A B C D . . . . . T . . . .

Searching for a number in an array

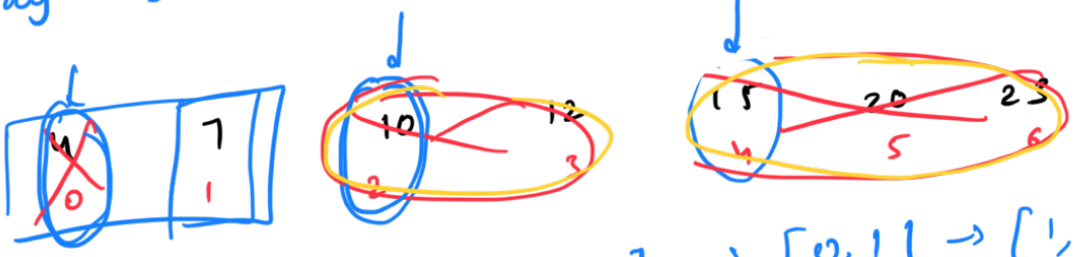
1) Array is not sorted

1) Sort and apply the above process
↓
O(n log n)

2) Linear Search : O(n)

2) Array is sorted.

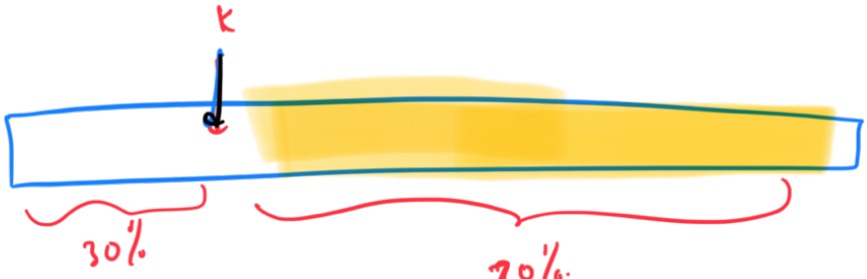
A :



Search Space: [0, 6] → [0, 3] → [0, 1] → [1, 1]

Sub array ⇒ Start index and End index

What would be the best strategy for picking thy random dictionary



Best Case: Discard 70% of search space

Worst Case: Discarding 30%

Worst Case:

Maximize the search space to be discarded even in the worst case.

Best: Discard 60%

worst: Discard 40%

→

Best: 50%

Worst: 50%

Divide the array into 2 equal halves at every iteration

Detn: divide the search

the process where we make a decision to go either to the left or right and discard the other half.

Search Space (Array) into 2 equal halves,

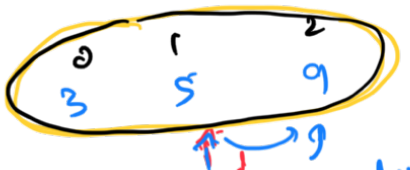
When can we apply binary Search?

X 1) Array is Sorted. ✗ (Blocked by Recruiter)
(Some sort)

1) Inherent order in the collection
(Not necessarily sorted)

2) Exploit this ordering to result half of the possibilities by doing exactly one comparison

Question: Search for an element in an array



EA = [3, 5, 9, 12, 19, 26, 29, 35, 39]

Target = 12

Search Space

low	high
0	8

F(12) = 3

F(15) = -1

1) If our mid lands on {3, 5, 9}

2) Go Right : low = mid+1.

if(arr[mid] < Target)

low = mid+1

2) If our mid lands on {19, 26, 29, 35, 39}

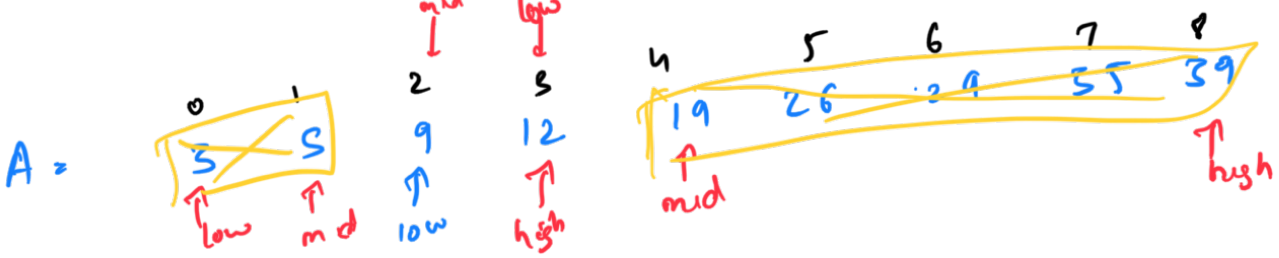
if(arr[mid] > Target)

high = mid - 1

3) // our mid lands on Target

if(arr[mid] == target)

return mid.



T = 12

low	high	mid
0	8	4
0	3	1
2	3	2
3	3	3

Action

high = mid - 1 = 3

low = mid + 1 = 2

low = mid + 1 = 3

Done : Return 3:

Iterative

```
int binarySearch( arr, N, target) {
    low = 0, high = n-1;
    while ( low <= high ) {
        mid = (high + low) / 2;
        if( Arr[mid] == target) return mid.
        if( Arr[mid] < target) low = mid+1;
        else: high = mid-1;
    }
    return -1;
}
```

Recursive code (Lecture Notes)

→ mid = (high+low)/2

Time Complexity:

Search space was n elements

Step 1: $\frac{n}{2} = \frac{n}{2^1}$

Step 2: $\frac{n}{4} = \frac{n}{2^2}$

Step 3: $\frac{n}{8} = \frac{n}{2^3}$

Step K: $= \frac{n}{2^k} = 1$

$n = 2^k$

$\log_2 n = k \log_2 2$

$k = \log_2 n$.

$k = \log n$

T.C: $O(\log n)$

$T(n)$ = Time for searching in an array of size N

$T(n) = O(1) + T(\frac{n}{2})$

$T(n) = T(\frac{n}{2}) + O(1)$

Master's theorem

$O(\log n)$

Question: Given a sorted array of distinct elements which contains duplicates and a target, Find the frequency of this element

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A =	-5	-5	-3	0	0	1	1	1	2	5	5	5	5	10	10

left, right

T = 5

1 : 3

5 : 4

10 : 2

3 : 0

$p_1 = 9$

$p_2 = 12$

$p_2 - p_1 + 1$

$12 - 9$

#

Brute Force:

```
for(i=0; i<n; i++)
    if(arr[i] == target)
        count++;
```

T.C: $O(n)$

S.C: $O(1)$

Approach 2:

-> Do a binary search to find any occurence of target

Time complexity:

A = 2 2 2 2 2 2 2 2 2

(1) — mid — (4)

$\frac{1}{4}$ $\frac{3n}{4}$

Target = 2

T.C: $O(n)$

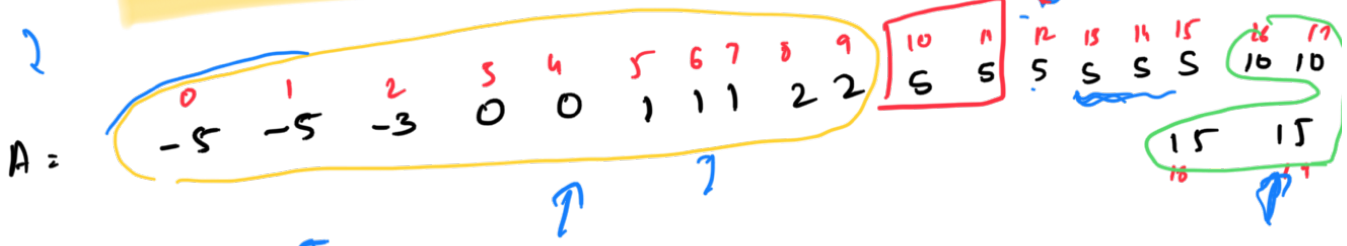
Approach 3 :

→ Find the first occurence of 5 : P_1

→ Find the last occurence of 5 : P_2

ans = 12

Ans = $P_2 - P_1 + 1$



Target = 5

1) If you land on {-5, -5, -3, 0, 0, 1, 1, 1, 2, 2}
Go Right => low = mid + 1

2) If you land on {10, 10, 15, 15}
Go Left => high = mid - 1

arr[mid], target

3) If you land on {5, 5, 5, 5, 5, 5}
Update the answer,
Go Left => high = mid - 1

Last occurence (Go Right => low = mid + 1)

```
int FirstOccurence ( Arr, n, target ) {
    low = 0, high = n-1;
    p1 = -1;
    p2 = -1;
    while (low <= high) {
        mid = (high + low)/2;
        if( arr[mid] < target) low = mid+1;
        else if (arr[mid] > target) high = mid-1
        else {
            p1 = mid;
            high = mid - 1;
        }
    }
}
```

-2

p_2 = mid
low = mid + 1

return

}

(p2 - p1 + 1) as the answers.

$p_2 = -1$

$p_1 = -1$

$p_1 - p_1 + 1$

$(-1) - (-1) + 1$

$= +1 + 1 = \boxed{1}$

$p_2 = -2$

$p_1 = -1$

$(-2) - (-1) + 1$

$= -2 + 1 + 1 = \boxed{0}$

if($p_1 == -1$ && $p_2 == -1$)

return 0;

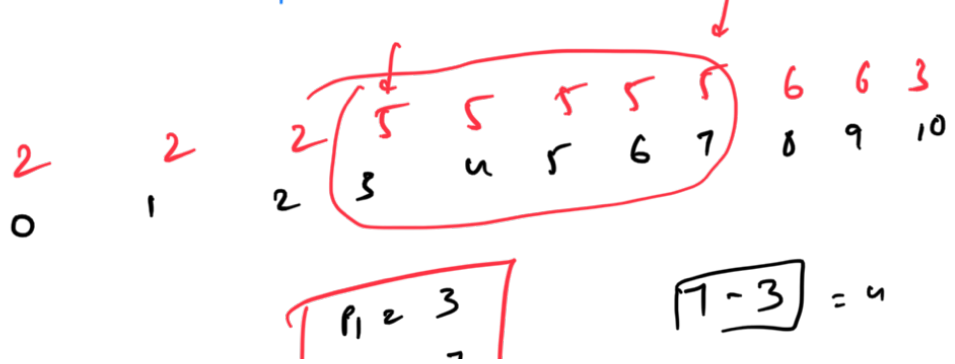
1) If we land on invalid part: we go towards the valid part.

2) If we land on valid part, update the answer, and search for a better answer.

1) Find fst occurence (logn)

2) " Last occurn (logn)

T.C.: O(logn)



$p_1 = 3$

$7 - 3 = 4$

$P_2 = 1$

$P_2 - P_1 = 4$

arr[mid] < arr[mid-1]

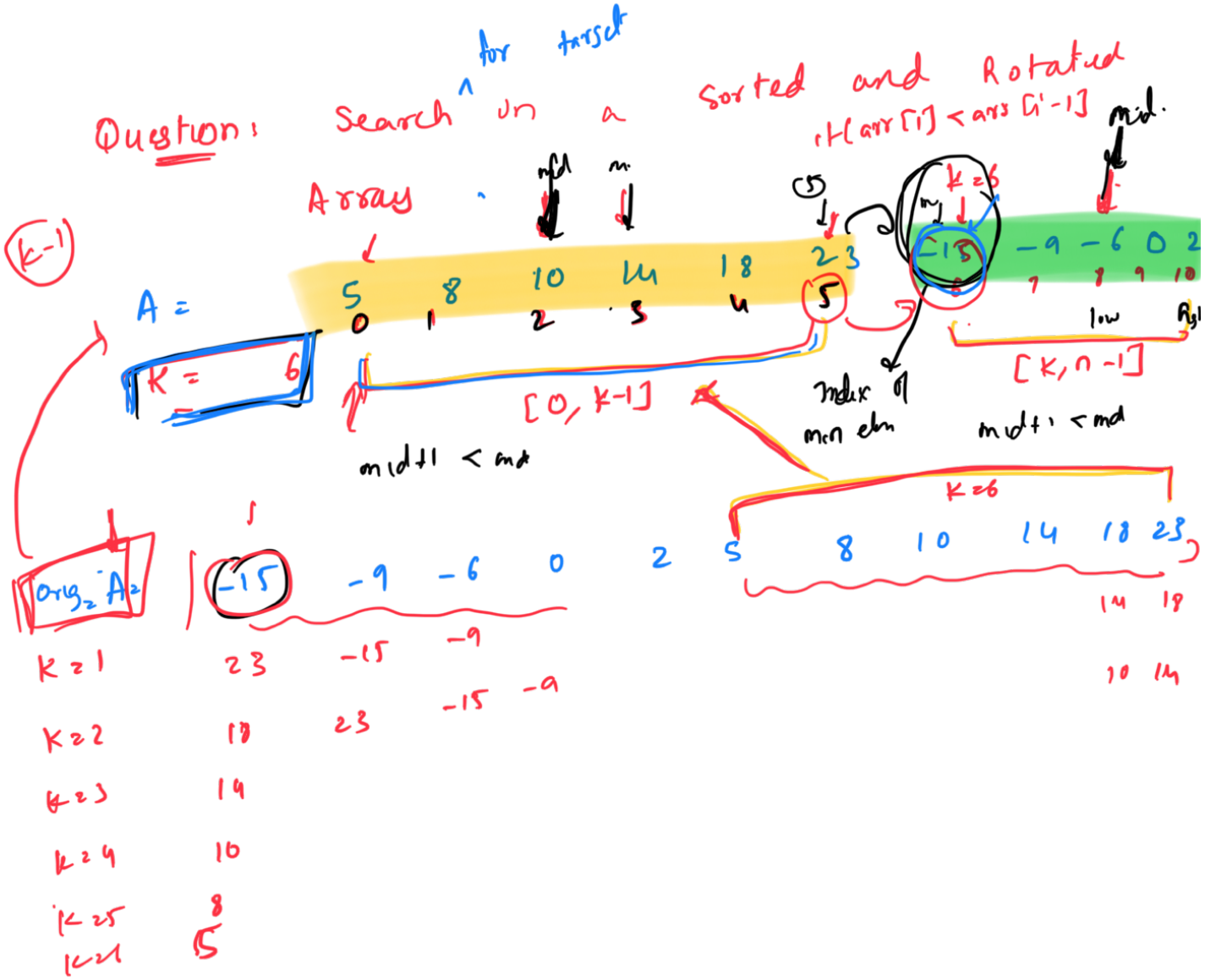
[3, 7]

[3 . . . 7-1]

3, 4, 5, 6

Question: Search for target in a Sorted and Rotated Array.

if [arr[i] < arr[i-1]



Brute Force :

Linear Search : O(n)

Approach 2 :

Binary Search on 2 Sorted Arrays

Arr1 : [0, K-1]

Arr : [K, n-1]

BS (arr, 0, K-1, target)

BS(arr, K, n-1, target)

T.C : O(log n)

What if K is not given

Approach 1 :

Iterate and find the first element which is less than its previous.

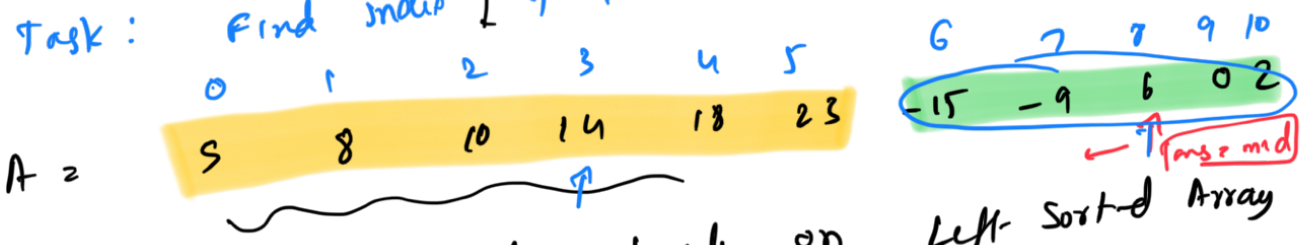
T.C : O(n) + O(log n) = O(n)

Approach 2 :

Find K in effient time

K => Index of 1st elements of the Right sorted Array (Index of min element of Right sorted Array

Task : Find index of 1st element



1) If mid Pointer lands on Left Sorted Array

Go Right => low = mid+1

2) If mid pointer lands on Right Sorted Array

{-15, -9, 6, 0, 2}

Update the answer.

Go Left => high = mid-1

-> How do we know we are on Left or Right Sorted Array

5 8 10 13 14

k = 0 : 3 5 8 10 13

k = 1 : 14 3 5 8 10

k = 2 : 13 14 3 5 8

k = 3 : 10 13 14 3 5

k = 4 : 8 10 13 14 3 5

-> All elements of Left sorted Array are greater than elements of Right Array

=) All elements of Left sorted Array are ≥ 5

=) All elements of Right sorted Array < 5

if(arr[mid] < arr[0])

RIGHT SORTED ARRAY

if(arr[mid] ≥ arr[0])

LEFT SORTED ARRAY

int rotationFactor(Arr[], N,){

low = 0, high = N-1;

while(low ≤ high){

mid = (high + low)/2;

-> if(arr[mid] ≤ arr[0]){

k = mid;

high = mid-1;

-> } else {

low = mid+1;

}

arr[mid] = 5

arr[0] = 5.

}

1) Find k → O(log n)
2) search(0, k-1) → O(log n)
3) search(k, n-1) → O(log n)

3 Binary search calls

if(arr[0] ≤ target) {
BS(0, k-1); (Left sorted Array)

}
else {
BS(k, n-1) (Right sorted Array)
}

Ex:

0	1	2	3	4	5	6
5	1	5	5	5	5	5

k, mid, low, high

low = 0, high = 6
mid = 3

arr[mid] = 5
arr[0] = 5

low = 4, high = 6

Question: → Given an array of elements, all elements repeat twice except 1 element.
→ Repeated elements are adjacent to each other.
→ Find the non-repeating element.

A: 3 3 1 1 4 6 6 8 8

Approach1:

XOR of all elements : O(n)

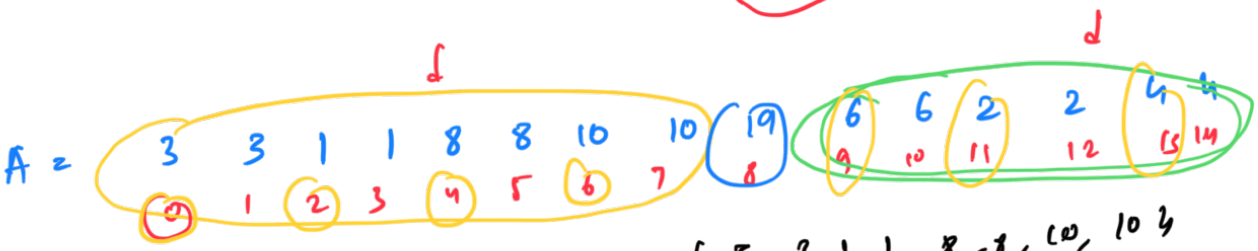
Approach2:

if (a[mid] == a[mid+1])

high = mid - 2

else

low = mid + 1



1) If mid lands on { 3, 3, 1, 1, 8, 8, 10, 10 }

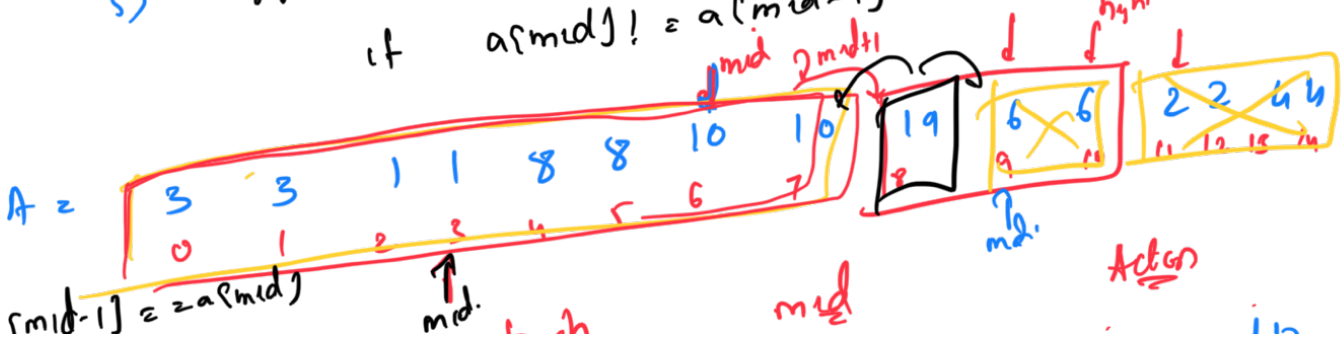
Go Right 2) low = mid + 1

2) If mid lands on { 6, 6, 2, 2, 4, 4 }

Go Left 2) high = mid - 1

3) If mid lands on 19

if a[mid] != a[mid-1] && a[mid] != a[mid+1]



a[mid-1] == a[mid]

Action

Low	high		
0	14	7	low = mid+2.
8	14	11	high = mid-1
8	10	9	high = mid-1.
8	8	8	
8			

```
int uniqueElement(arr, N) {
    low = 0, high = n-1;
    // Base case.
    while(low <= high) {
        mid = (low+high)/2;
        if(arr[mid] != arr[mid-1] &&
           arr[mid] != arr[mid+1])
            return arr[mid];
        // update mid to first occurence
        if(arr[mid] == arr[mid-1]);
            mid = mid-1.
        if(mid%2 == 0)
            low = mid+2;
        else
            high = mid-1;
    }
}
```

(mid%2 == 1)

→ arr[mid] != arr[mid-1] && arr[mid] == arr[mid+1]

mid-1, mid, mid+1.

Base Cases

1) Array is single Element Array

A = [3]

if(n == 1) return arr[0]

mid.

2) 6 5 3 1 1

arr[mid-1]

arr[-1]

if(arr[0] != arr[1]

return arr[0];

3) 3 3 1 1 4 4 6

mid

arr[mid+1]

if(arr[n-1] != arr[n-2])

return arr[n-1];

T.C: O(logn)

low = 0

high = 8

high = mid-1

mid = 4

Low = 0

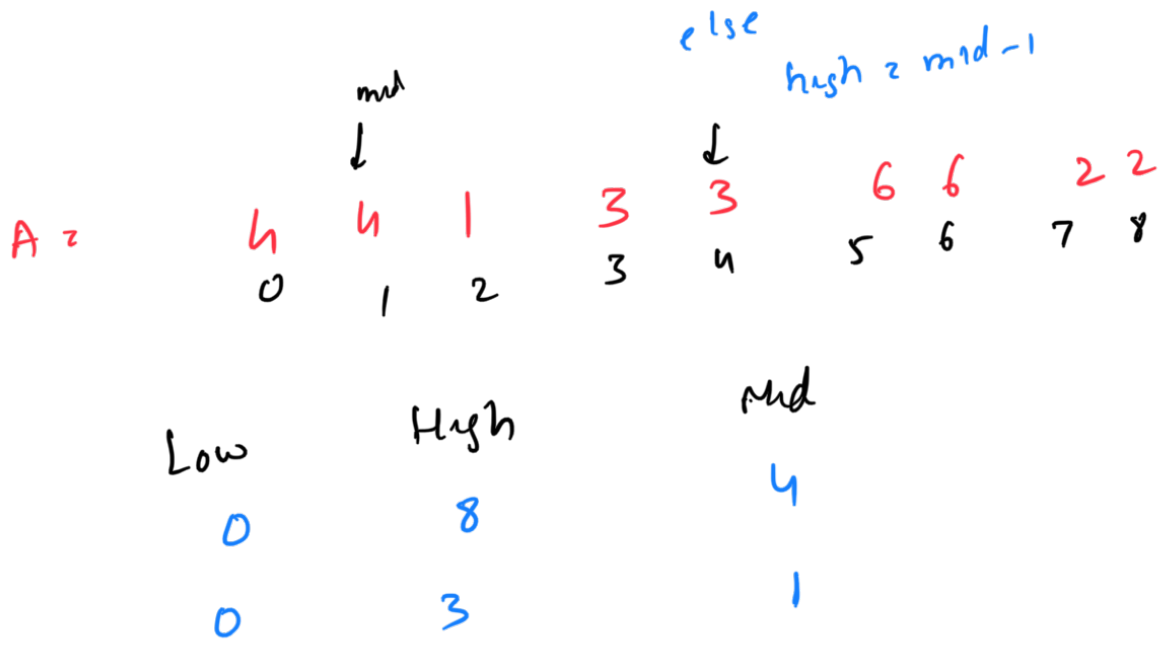
High = 8

mid = 4

high = mid-1

if(arr[mid] == arr[mid+1]

low = mid+1



Recursive code:

```
int BSR(int arr[], int low, int high, int val{
      if(low > high) return -1;

      mid = (high + low)/2

      if(arr[mid == k)
            return mid;

      // We are on right side; we should go left
      if(arr[mid] > val)
            return BSR(arr, low, mid-1, k);

      // We are on left side; go right
      else
            return BSR(arr, mid +1, high, k);

}
```