⊛ Binary Search Algorithm:

- It is used for sorted arrays

arr = [2, 4, 9, 10, 12, 14, 18, 19]

→ ascending.

arr 2 = [19, 12, 6, 5, 3, 2, −8, −16]

→ descending.

In linear search maximum comparision in the worst case is N (no of elements in array)

0 1 2 3 4 5 6 7 8 9

arr = [2, 4, 6, 9, 11, 12, 14, 20, 36, 48]

M

Step 1: find the middle element.

Step 2: Check if the target element is greater than middle element Search in right or else search in left.

Step 3: If the middle element = target element then the answer is found

target = 36

Step 1: find the middle element

middle element = 45 = 4th index

i.e ⟶ 11

$m = \frac{s+e}{2} \Rightarrow \frac{start + end}{2}$

Step 2:

target element is in the right side of M

0	1	2	3	4	5	6	7	8	9
2	4	6	9	11 (M)	12	14	20	36	48

arr = [12, 14, 20, 36, 48]

5	6	7	8	9
12	14	20 (M)	36	48

$M = \frac{s+e}{2} = \frac{5+9}{2} = \frac{14}{2}$ = 7th index

target = 36 > 7th index = 20

0	1	2	3	4	5	6	7	8	9
2	4	6	9	11 (M)	12	14	20	36	48

8	9
36	48

$M = \frac{8+9}{2} = \frac{17}{2}$ = 8th index

target = 36 8th index = 36

36 = 36 target found

$$arr = [\underset{0}{2}, \underset{1}{4}, \underset{2}{6}, \underset{3}{9}, \underset{4}{\underset{M}{11}}, \underset{5}{12}, \underset{6}{14}, \underset{7}{20}, \underset{8}{36}, \underset{9}{48}]$$

target element = 12

$$Middle\ element = \frac{0+9}{2} = 4.5 = 4th\ index$$

target = 12 $\underset{(m+1)\ first}{>}$ index = 11

$$arr = [\underset{s}{\underset{5}{12}}, \underset{6}{14}, \underset{M}{\underset{7}{20}}, \underset{8}{36}, \underset{e}{\underset{9}{48}}]$$

$$M = \frac{s+e}{2} = \frac{5+9}{2} = 7th\ index$$

target = 12 $\underset{M-1\ (first)}{<}$ index = 20

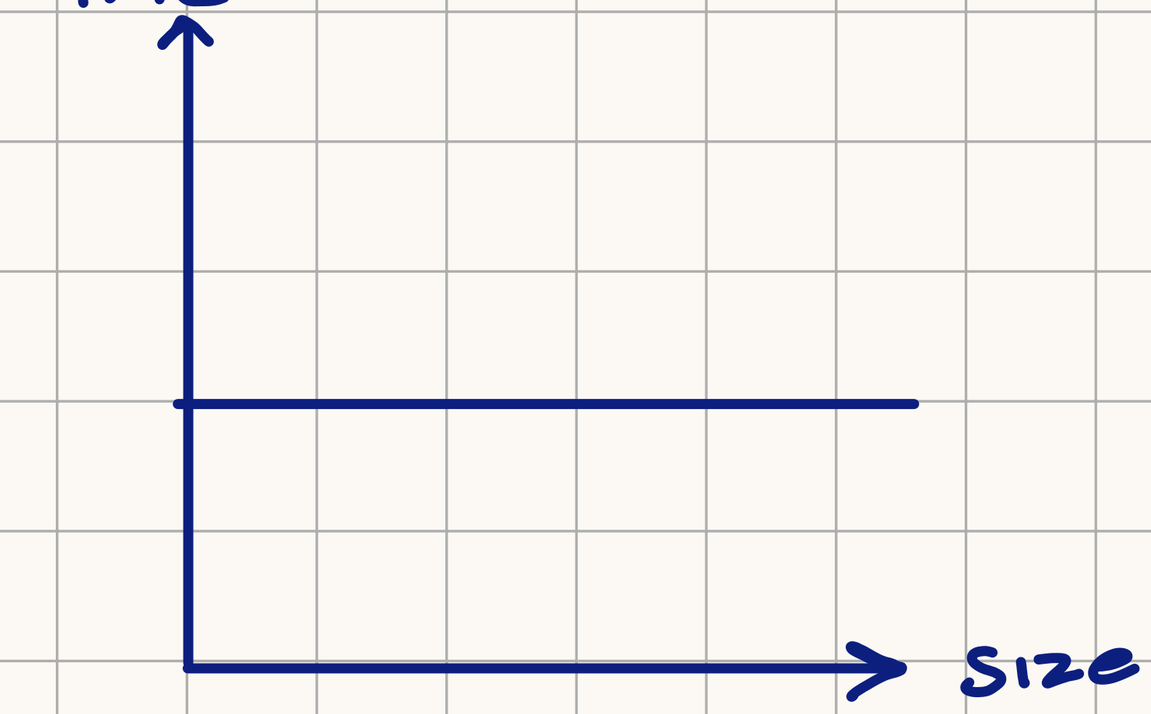
$$arr = [\underset{5}{12}, \underset{6}{14}]$$

$$M = \frac{5+6}{2} = \frac{11}{2} = 5.5 = 5th\ index$$

target = 12 = index = 12

hence target element is at 5th index

best case complexity is constant

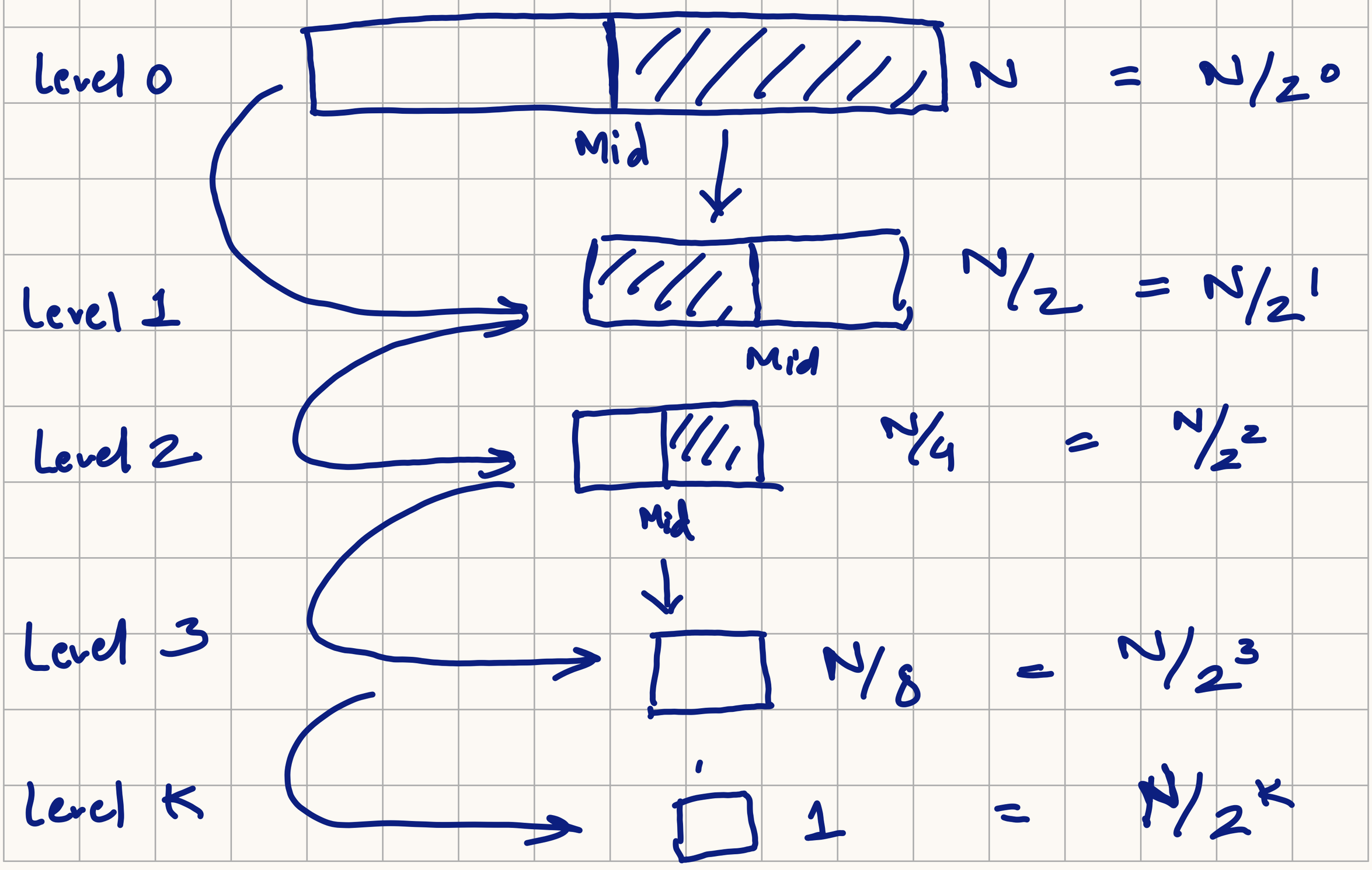


Best case

O(1)

* Why Binary Search ?

a) Find the max number of such comparisio
in the worst case



$$\frac{N}{2^k} = 1 \Rightarrow N = 2^k$$

$$\log(N) = \log(2^k)$$

$$\log N = k \log 2$$

$$k = \frac{\log N}{\log 2} \Rightarrow k = \log_2 N.$$

k is the total number of levels in the worst case.

So the total number of levels or total number of comparisions in the worst case is log N. where N is the size of array.

Total comparision in worst case = log N

That means if we compare linear search with binary search we get.
Let us suppose there are 1,00,000 element

Linear search	Binary Search
no. of comparisions:	no. of comparisi $\log_2(100000)$
1,00,000 compares	= 17 comparisions

Complexity of binary search is big O(logN)

Java program for binary search:

```
public class Main {
    public static void main (String[] args){

        int [] arr = {-18, -12, -4, 0, 2, 3, 4, 15,
                      16, 18, 22, 45, 89};
        int target = 22,
        int ans = binarySearch (arr, target);
        System out println (ans);
}

 Static int binarySearch (int[] arr, int target)
        int start = 0,
        int end = arr length-1;
        while (start <= end) {
// int mid = (start + end)/2; // might exceed int
        int mid = start + (end-start)/2;   limit
        if (target < arr[mid] {
                end = mid-1;
         }else if (target > arr[mid] {
                start = mid+1;
        } else {
                return mid,
                }
        }
        return -1;
    }
}
```

* Order agnostic binary search

s m e

arr = [90, 75, 18, 12, 6, 4, 3, 1]

0 1 2 3 4 5 6 7

target = 72

If target > middle element ⇒ LHS

↳ e = m - 1

target < middle ⇒ RHS

↳ s = m + 1

arr = [1, 3, 5, 6, 9, 12, 14, 20, 33]

if s > f ⇒ increasing order sorted

else ⇒ decreasing order sorted.

Code:

```
public class OrderAgnosticBS {
        public static void main (String[] args) {

int [] arr = {-18,-12,-4, 0,2,3,4,15, 16,18,22,45,89};
    int target = 22;
    int ans = orderAgnosticBS(arr, target);
     System out.println(ans);

}

   Static int orderAgnosticBS (int[] arr, int target) {
     int start = 0;
    int end = arr.length-1;
    // find whether the array is sorted in ascending or
       descending
    boolean isAsc = arr[start] < arr [end];

    while (start <= end) {
        int mid = start + (end- start)/2;
      if (arr[mid] == target)
          return mid;
```

```
if (isAsc) {
  if (target < arr[mid]) {
        end = mid-1;
  } else if (target > arr[mid] {
        start = mid+1;
   }
   }
      else {
          if (target > arr[mid]){
             end = mid-1,
        } elseif (target< arr[mid]) {
          start = mid+1;
      }
  }
    return -1;
} }
```

Binary Search Interview Problems.

Q) Ceiling of a number

arr = [2, 3, 5, 9, 14, 16, 18]

// sorted array can be seen hence binary search will be applied.

target number = 15.

* Ceiling = Smallest element in an array greater than equal to target element

Example: ceiling for target element = 14 in same array

ceiling (arr, target = 14) = 14

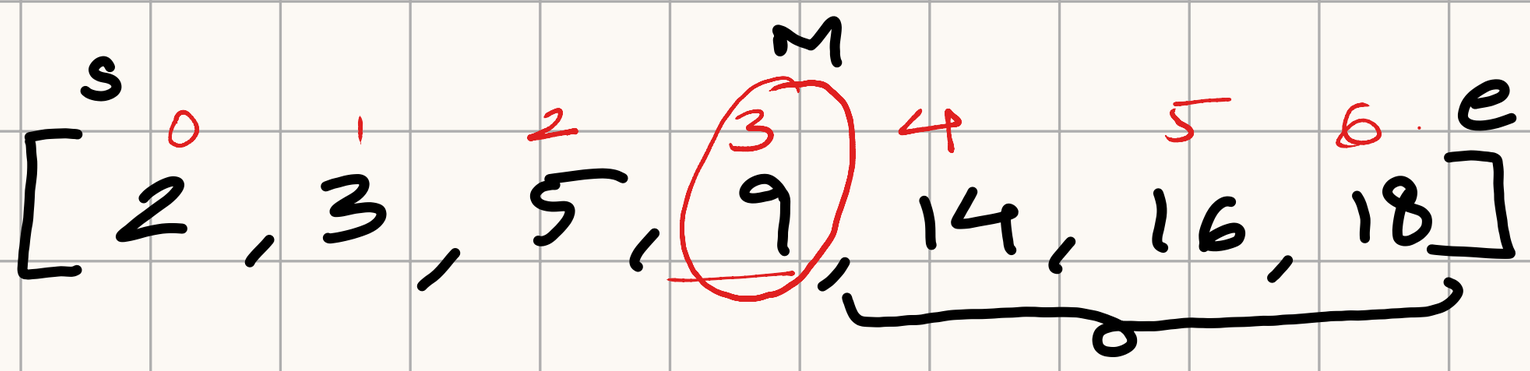
ceiling (arr, target = 15) = 16

Ceiling (arr, target = 4) = 5

ceiling (arr, target = 9) = 9

arr = [2, 3, 5, 9, 14, 16, 18]

target = 15



s = 4 e = 6

$\frac{4+6}{2}$ = 5th index

↓

16

e	s
4	5
14	16

6

18

s target e

⇓

e target Start

→ answer

→ // condition for while loop violated

Lets talk about start and end pointer

* Find a number ≥ target

Start ≤ end ⇒ when while loop breaks

Start = end + 1

hence next big number greater than target is start

★ Floor of a number

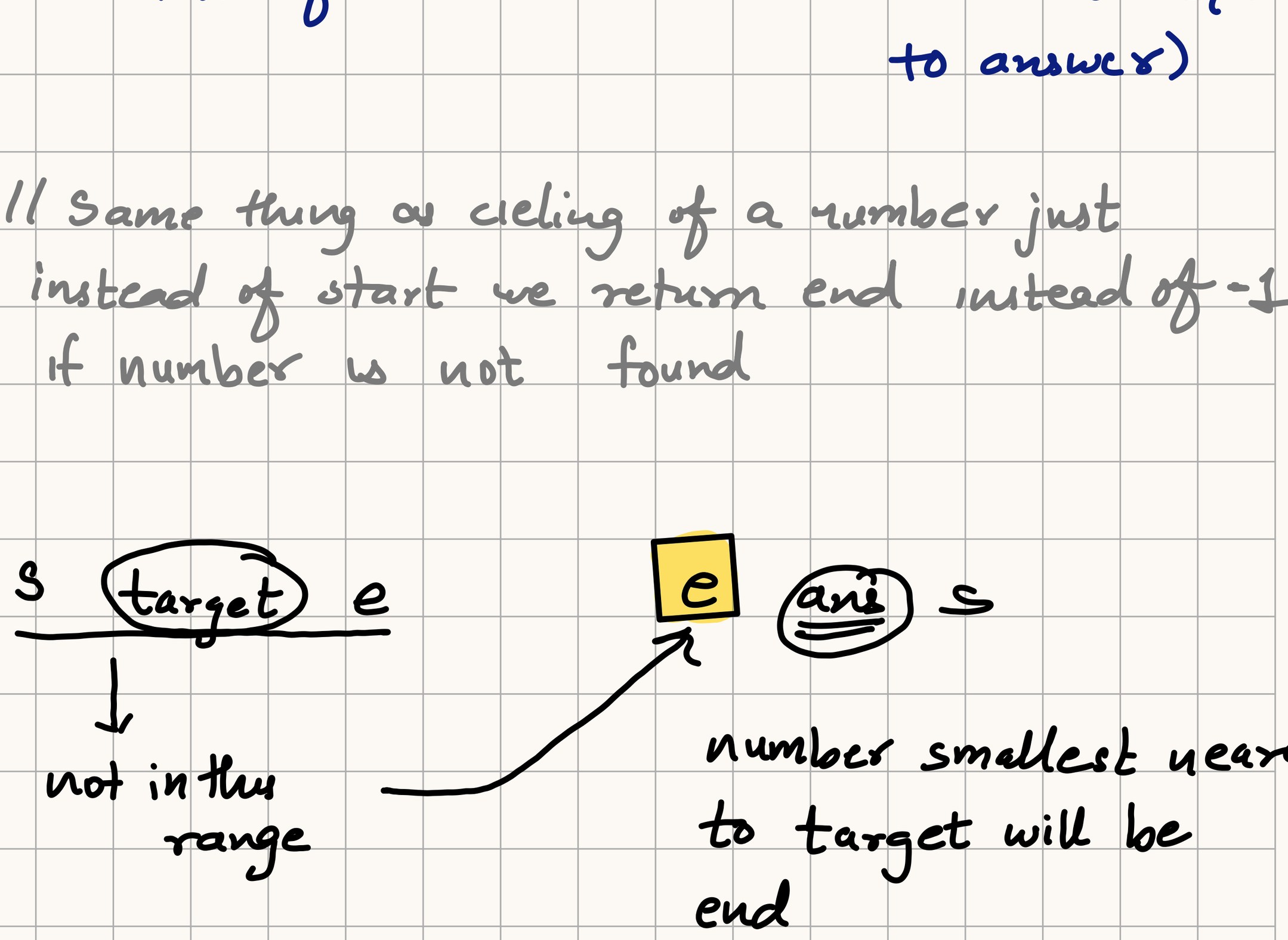
Q) Find the floor of a number

floor = greatest number that is smaller or equal to target number

arr = [2, 3, 5, 9, 14, 16, 18]
0 1 2 3 4 5 6

floor of 15 ⟶ 14 (smaller or equal to answer)

// Same thing as cieling of a number just instead of start we return end instead of -1 if number is not found



when condition is voilated

$S = e + 1$ $\quad s > e$

⊙ Code for cieling of a number.

```
public class Main {
 public static void main (String [] args) {
int [] arr = {2, 3, 5, 9, 14, 16, 18};
int target = 15;
int ans = ceiling (arr, target),
    System out.println (ans),
}


static int ceiling (int [] arr, int target){

        if (target > arr [arr length-1]) {
            return -1;
         }

        int start = 0,
        int end = arr length-1 , // indices of array elements

        while (start <= end) {
           int mid = start + (end-start)/2,
```

```
        if ( target < arr[mid] ) {
                end = mid - 1,
          }
        else if (target > arr[mid]) {
                start = mid + 1,
            }
            else {
                return mid,
            }
        }
        return start;
    }
}
```

⊙ Code for floor of a number

```
public class main {
    public static void main (String [] args){
    int [] arr = {2, 3, 5, 9, 14, 16, 18},
    int target = 3,
    int ans = ceiling (arr, target),
   System out println(ans);
}
```

```
Static int floor (int[] arr, int target){
    int start =0;
    int end = arr length -1; // indices of array element
    while (start >= end){
            int mid = start +(end-start)/2;
           if (target < arr [mid] {
                end = mid -1,
        }
    else if (target > arr[mid] {
            start = mid+ 1;
        }
    else {
        return mid;
        }
        }
        return end,
    }
}
```

⊙ Find smallest letter greater than target (LC-744).

★ Exact same approach for ceiling of a number

★ Ignore the target = what we are looking for

★ arr = ['c', 'd', 'f', 'j'], target = 'j'
(0) (1) (2)

When you find a case where no element is larger than target, return first element

condition violated : start = end + 1;

↳ length of array = N

return S % N

if S == N

return 0,

2 % 4 = 2 // rct start

4 / 4 = 0

⊙ Find first and last position of Element in sorted Array (LC-34)

arr = [5, 7, 7, 7, 7, 8, 8, 10]
indices: 0, 1, 2, 3, 4, 5, 6, 7

first index of 7 ↓ (index 1)

last index of 7 ↓ (index 4)

Answer → [1,4]

target = 7

* find first occurence of 7.

s m e

5, 7, 7, 7, 7, 8, 8, 10

Ans

{ binary search target = 7.

Ans = 3 is a possible answer but there is a chance that more 7 are occuring before 1 index 3

for first occurence of 7 ⟶ e = m - 1

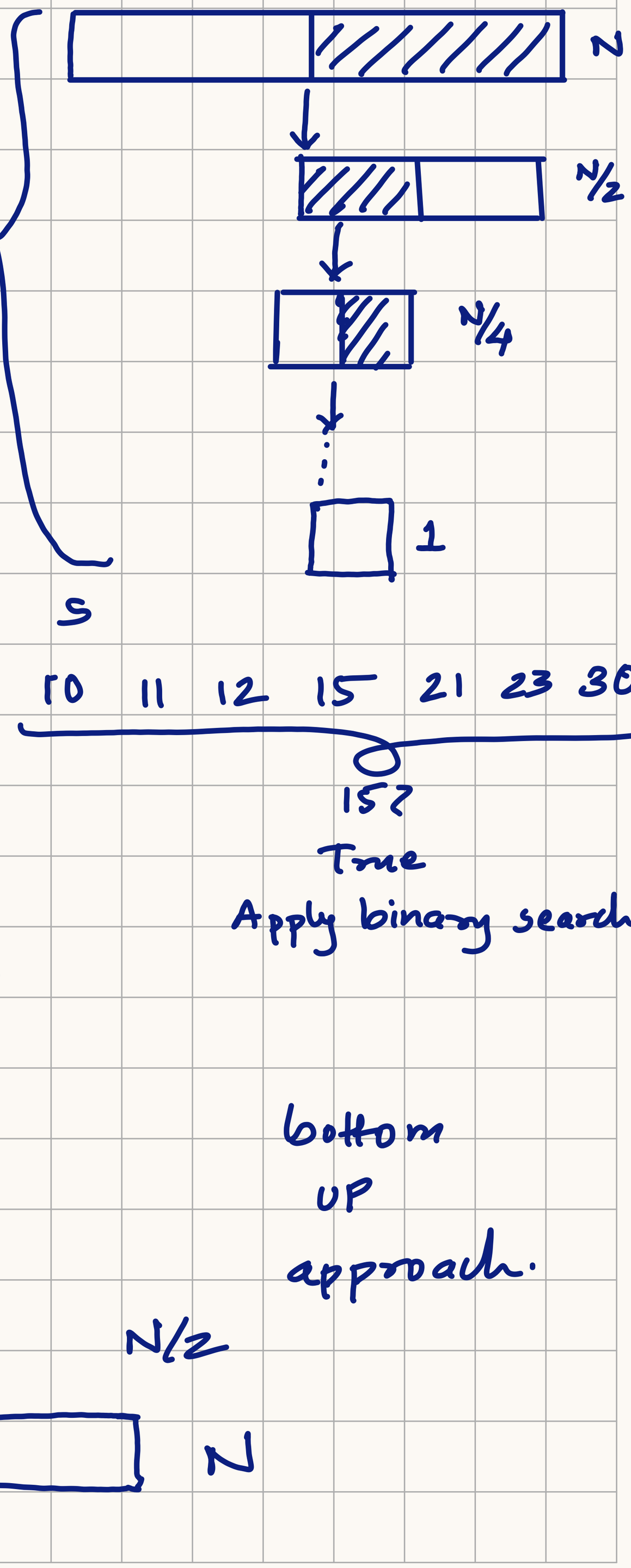
for last occurence of 7 ⟶ s = m + 1

⊙ Find position of an element in a sorted array of finite numbers

s — 1st chunk — e — s — 2nd chunk — e

arr = [2, 3, 5, 6, 7, 8, 10, 11, 12, 15, 20, 23, 30]

target = 15



⊙ Code for above logic

```
public class Main {
 public static void main (String[] args) {

int[] arr = { 3,5,7,9,10,90,100,130,140,160,170 };
int target = 10;
System.out.println (ans( arr, target));

 }

Static int ans (int[] arr, int target) {
        // first start with a box of size 2
        // first find the range
 int start = arr[0];
 int end = arr[1];
 // condition for the target to lie in the range
while (target > arr[end]){
     int newStart = end + 1; // this my new start.
     //double the box value
   // end = previous end + sizeofbox * 2
     end = start + (end - start +1)*2,
     start = newStart;
}
 return binarySearch (arr, target, start, end),
```

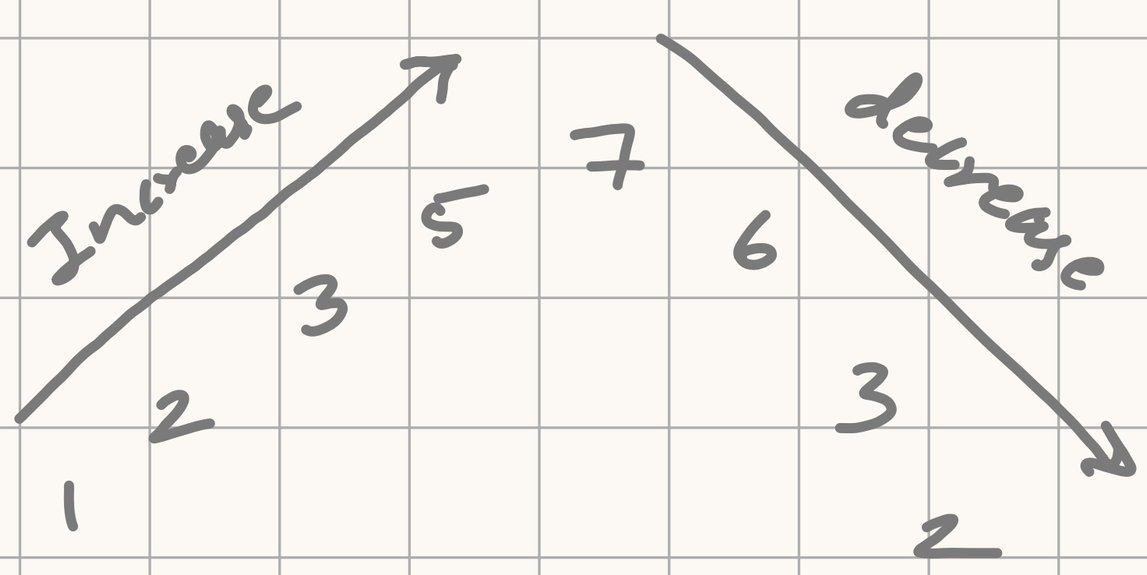
```
static int binarySearch(int[] arr, int target, int
start, int end){

while (start <= end) { // common in all binary
    int mid = start + (end - start)/2,          search
    if (target < arr[mid]){
        end = mid-1,
    }else if (target > arr[mid]){
        start = mid+1;
    } else {
        return mid,
    }
} return -1;
}
```

⊙ Peak index of mountain array (bitonic array)

LC · 852

example of bitonic array

arr = [1, 2, 3, 5, 7, 6, 3, 2]



Q) Find peak in mountain array

arr = [0, 2, 1, 0]

2 → answer (peak)

0 1 0

→ Linear search can be used but that has no good time complexity

→ binary search will be used for most optimized solution.

arr = [1, 2, 3, 5, 6, 4, 3, 2]
(0 1 2 3 4 5 6 7)
s m e

Possibility.

① If element in middle is greater than element at middle + 1. This means that we are at decreasing part of the array

checking
s m e

⇒ if e[mid] > e[mid + 1]
⇒ We in descending part of array
⇒ Hence peak should exist in LHS of M+1.

② If middle element is less than middle element + 1 then we are in the ascending part of array

⇒ Start = mid + 1

checking
s m, m+1 e

③ When will loop break?

0 1 2 3 4 5 6 7
1, 2, 3, 5, 6, 4, 3, 2
s m e

6, 4
s e
m

6 (answer)
s, e
In the end both Start and end pointer point at greatest element

```
public class Mountain {
    public static void main (String[] args) {

    }
    public int peakIndexInMountainArray (int[] arr){
        int start = 0,
        int end = arr.length - 1,

        while (start < end) {
            int mid = start+(end-start)/2,
            if (arr[mid] > arr[mid+1]) {
                // You are in decreasing part of array.
                // this may be the ans, but look at left
                end = mid, // this is why end != mid-1
            } else {
                // You are in ascending part of array
                start = mid +1, // because we know that
                                // mid+1 is > mid element
            }
    }
```

```
        // in the end, start == end and pointing to the
           Largest number
        // start and end are always trying to find max
           element hence when they are pointing to
           just one element. that is the maximum one
      // at every point of time for start and end,
         they have the best possible answer till that
         time and if we are saying that only one item
         is remaining, hence cuz of above line that is
         the best possible answer
            return start, // or end
    }
}
```

⊙ Find in Mountain Array LC-1095

arr = [1 (0), 2 (1), 3 (2), 4 (3), 5 (4), 3 (5), 1 (6)] target = 3

Ans - ① Find peak element → 4th index

② Search in ascending array ⇒ (0, 4)

③ If not found, binary search in decreasing array ⇒ (4, 6)

⊙ Rotated Array

Normal sorted array

arr = [2, 4, 5, 7, 8, 9, 10, (12)]

↳ 1 rotation

After 1 rotation.

arr = [(12), 2, 4, 5, 7, 8, 9, (10)]

2nd rotation

After 2nd rotation.

arr = [10, 12, 2, 4, 5, 7, 8, (9)]

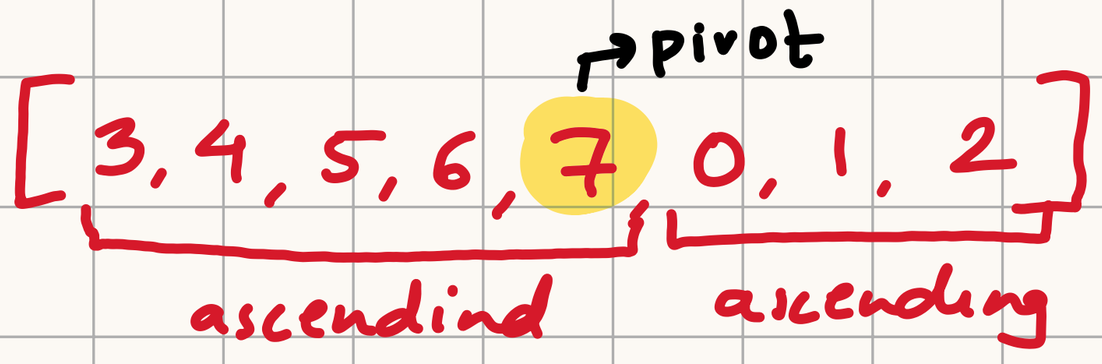
3rd rotation

After 3rd rotation.

arr = [9, 10, 12, 2, 4, 5, 7, 8]

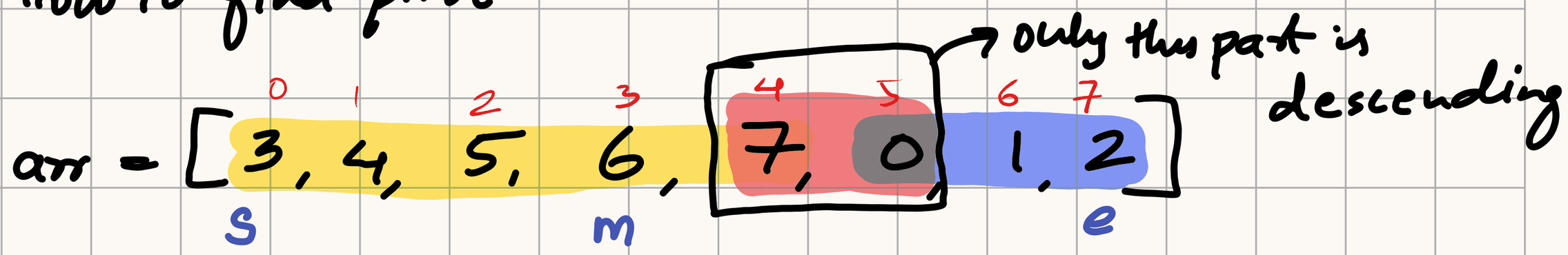
① Find the pivot in the array.

pivot ⇒ from where your next number are ascending



* Find pivot
* Search in first half ⇒ Simple binary search.
(0, pivot)
* Otherwise, search in second half (pivot+1, end)

☆ How to find pivot?

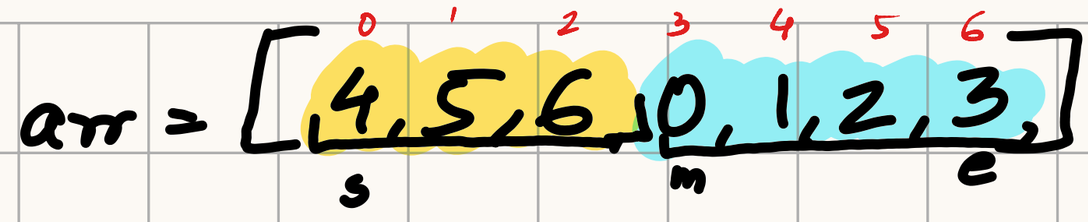


When you will find that mid > mid+1 element, i e pivot // Case 1

if mid element < (mid-1) element

i.e also answer // Case 2

Start element > mid-element // Case 3

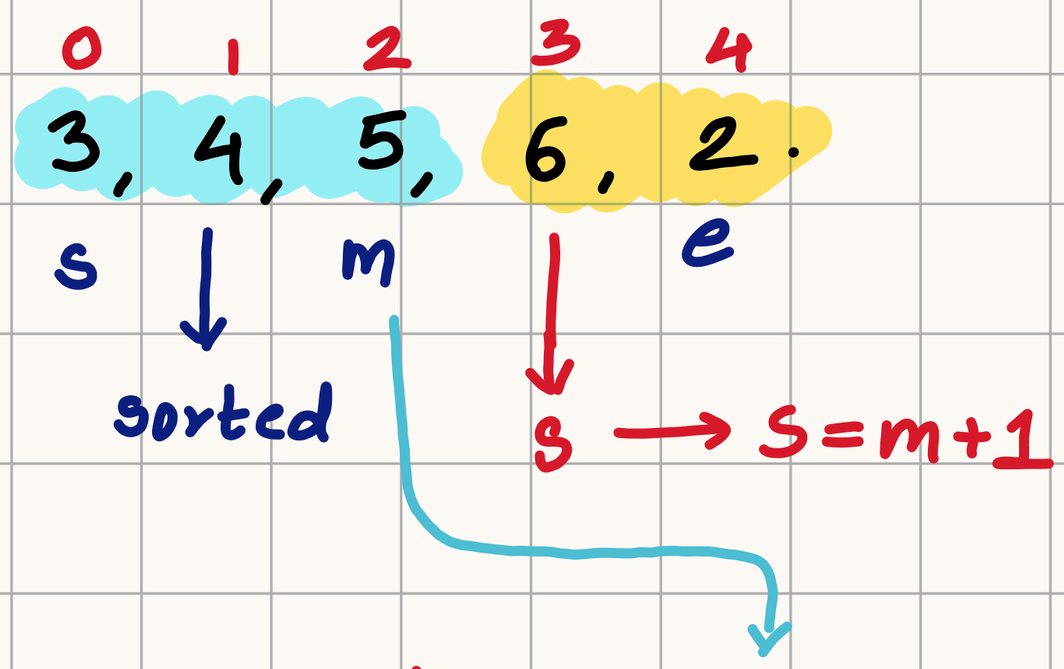


In this case, all elements from mid will be < start
Hence, we can ignore all these elements since we are looking for peak i.e largest element,

$e = mid - 1$

// Case 4. Start element is smaller than mid element

S element < mid element



Hence proved that bigger number lies ahead
Hence ignore mid and pivot $S = mid + 1$

arr = [4, 5, 6, 7, 0, 1, 2]
s p e

Case 1 · pivot element = target → Ans

Case 2 target > start element then search space is reduced to (s, p-1) because all numbers after pivot are smaller than start.

Case 3: target < start element

i.e we know that all elements from start till pivot are going to be bigger than target // target > 1

Search space = (pivot+1 till end)

⊛ Code to search in an Rotated Array [LC-33]

```
public class RBS {
        public static void main (String[] args){
            int[] arr = {4,5,6,7,0,1,2},
            System out println (findPivot(arr)),
    }

Static int search (int[] nums, int target){
        int pivot = findPivot (nums);
        //if you did not find a pivot, it means array is not
                                                    rotated
        if (pivot == -1){
        //just do normal binary search
       return binarySearch (nums, target, 0, nums length-1);
    }
        if (nums [pivot] == target){
            return pivot,
    }
        if (target >= nums [0] {
                return binarySearch (nums. target, 0, pivot -1),
          }
            return binarySearch (nums, target, pivot+1, nums
                                                    Length -1),
          }
Static int binarySearch (int[] arr, int target, int start :int end){
        while (start <= end) {
                int mid = start + (end - start) /2,
            if (target < arr [mid]) {
                end= mid - 1,
```

```
        } else if (target > arr[mid]) {
            start = mid + 1;
        } else {
            return mid;
        }
    } return -1,
}
```

```
static int findPivot (int[] arr){
    int start = 0,
    int end = arr.length - 1;
    while (start <= end) {
        int mid = start + (end - start)/2;
            // 4 cases
        if ( mid<end && arr[mid] > arr[mid + 1]){
            return mid;
        }
        if (mid > start && arr[mid] > arr[mid - 1]) {
            return mid - 1,
        }
        if ( arr[mid] <= arr[start] ) {
            end = mid - 1,
        } else {
            start = mid + 1;
        }
    }
    return -1;
```

Searching in Matrices

	0	1	2
0	18	9	12
1	36	-4	91
2	44	33	16

target = 91

Ans · [1,2]

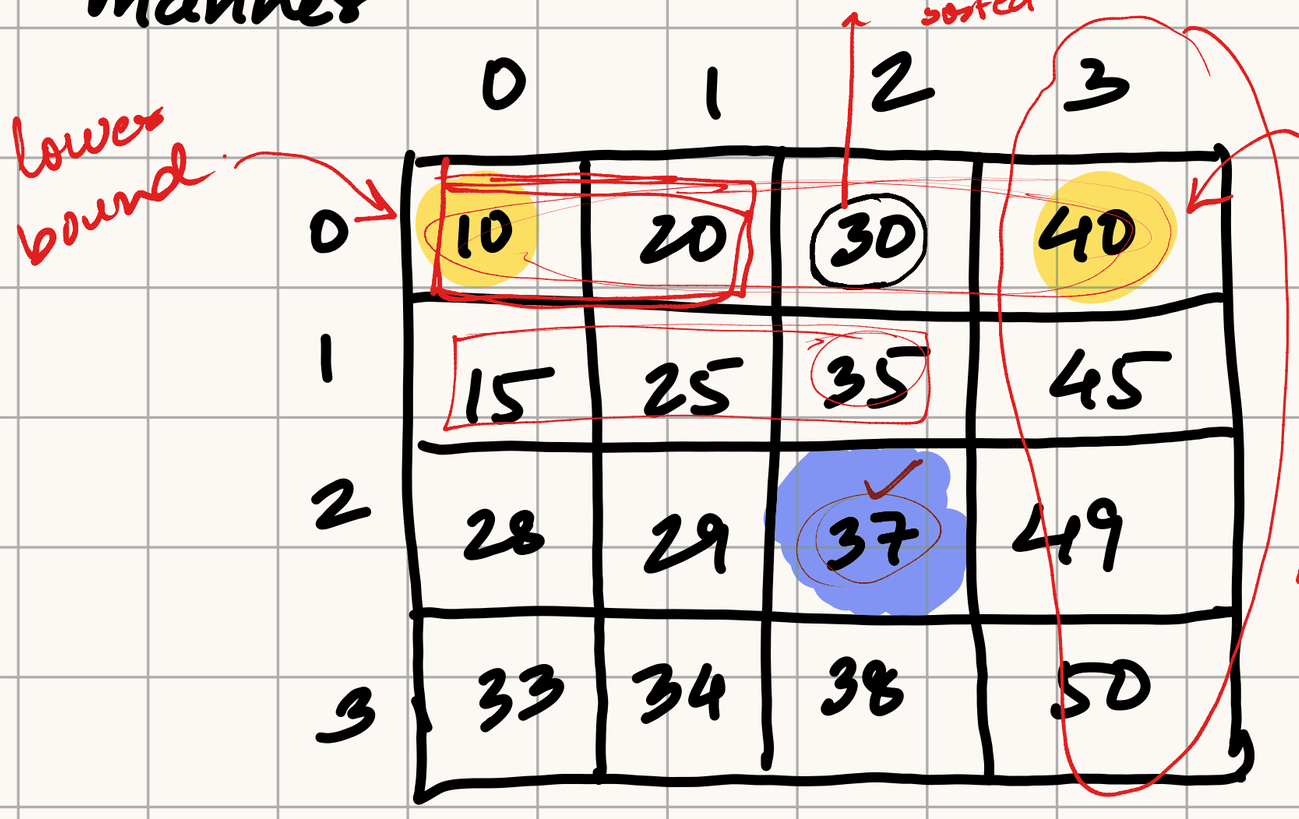
N * N = N^2

= $O(N^2$

O(N * M)

```
for row=0, r<n, r++.
    for col=0; col<n; c++.
        if arr[r][c]=target
return -1 //if not found
```

Q) Matrix is sorted in a row wise and columnwise manner



target = 37

Case 1. If the element is == target

↳ ans found

Case 2. If element < target

⇒

Case 3. If element > target

Time complexity ⇒ n + n = 2n.

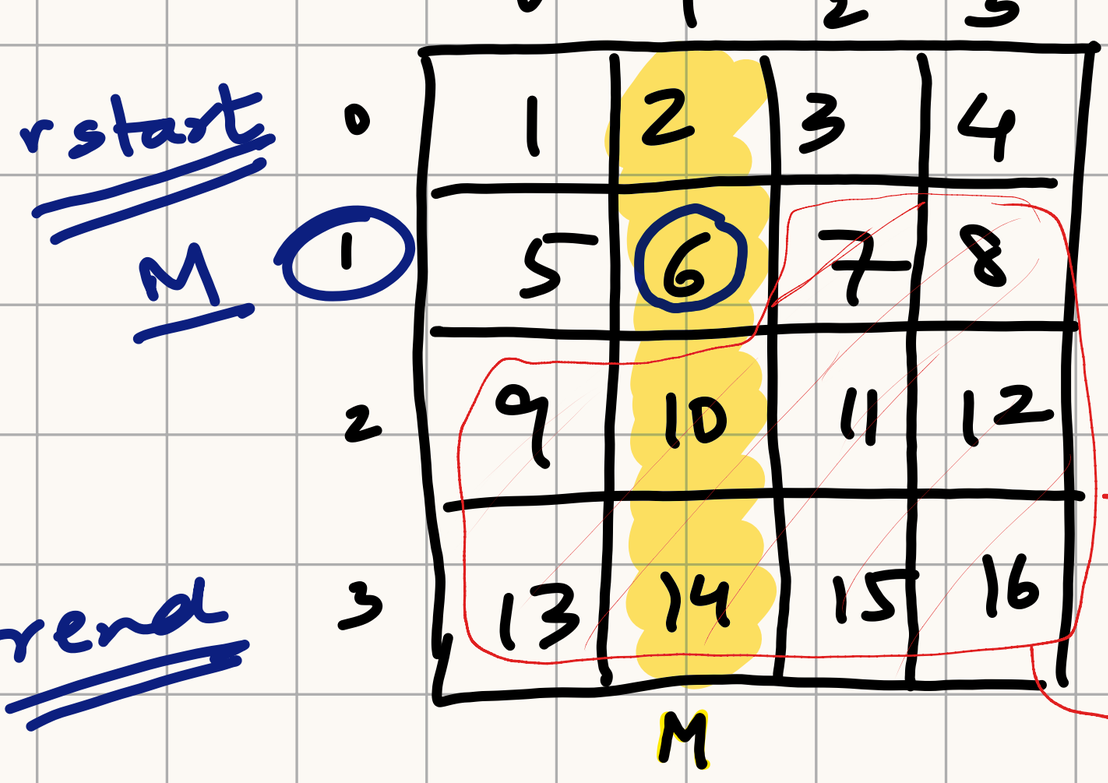
* Code for above problem.

```
public class RowColMatrix {
    public static void main (String[] args) {
        int [][] arr = {
                {10, 20, 30, 40},
                {15, 25, 35, 45},
                {28, 29, 37, 49},
                {33, 34, 38, 50}},
        int target = 37,
    System out println (Arrays to String (search (arr, target)));

static int[] search (int[][] matrix, int target){
        int r = 0,
        int c = matrix length-1;
        while (r< matrix length && c >=0) {
            if (matrix [r][c] == target) {
            return new int[][]{r, c};
        } if ( matrix [r][c] < target) {
                r++,
        } else {
                c--,
    } }
}
```

```
        return new int [] {-1,-1};
    }
```

Q) Search in a sorted matrix

target = 2



* Take middle coulumn & perform Binary search on it

-> > mid element = 6

6 > 2

-> > target = 2.

① If element == target

// ans

② If element > target

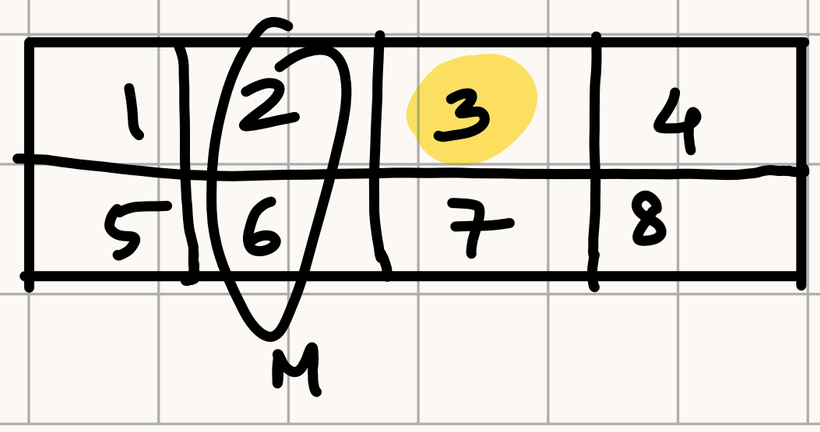
// ignore row after it

③ If element < target.

// ignore rows before it.

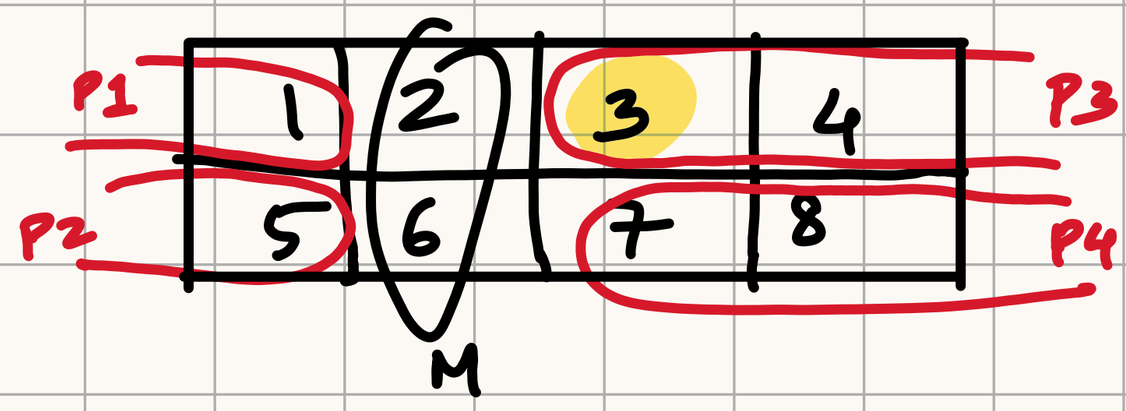
} reducing search space

In the end 2 rows are remaining target = 3



① check whether mid column you are at contains the answer [2,6]

② Consider 4 parts



Time complexity $\underline{O(\log(N) + \log(M))}$

* Code for above problem:

```
public class SortedMatrix {
 public static void main (String[] args) {

        int[][] arr = {{1, 2, 3},
                       {4, 5, 6},
                       {7, 8, 9}};
System.out.println(Arrays.toString(search(arr, 6)));

    }
  Static int[] search(int[][] matrix, int target){
      int row = matrix.length;
      int col = matrix[0].length, // be cautious, matrix
                                     may be empty.
      if (rows == 1){
          return binarySearch(matrix, 0, 0, cols-1, target);
      }
```

```
int rStart = 0;
int rEnd = rows - 1,
int cMid = cols/2,
// run loop till 2 rows are remaining
while (rstart < (rEnd-1)) { // while this is true
it will have more than two rows   int mid = rStart + (rEnd-rStart)/2
if (matrix[mid][cmid] == target){
    return new int[] {mid, cMid},
}
if (matrix[mid][cmid] < target){
    rstart = mid;
} else {
    rEnd = mid;
}
// now we have 2 rows
// check whether the target is in column of 2 rows.
if (matrix[rStart][cMid] == target){
    return new int[] {rstart + 1, cMid},
}
// otherwise search in 1st half
    if (target <= matrix[rstart][cMid-1]){
return binarysearch(matrix, rstart, 0, cMid-1, target),
}
if (target >= matrix[rstart][cMid+1] && target <= matrix[rstart][cols-1]){
rtn binarysearch(matrix, rstart, cMid+1, cols-1, target);
```

```
        }
        if (target <= matrix [rStart + 1] [cMid - 1]) {
return binary search( matrix, start +1 , cMid +1 , cols -1 , target);
        }
        else {
return binarysearch( matrix, rstart+1 , cMid+1, cols-1, target).
        }
```

```
Static int [ ] binarysearch (int [ ] matrix, int row, int cstart, int cEnd,
                                                                target) {
        while (cstart <= cEnd) {
        int mid = cStart + (cEnd - cStart) /2,
        if ( matrix [row] [mid] == target ) {
            return new int [ ] { row, mid};
          }
        if (matrix [row] [mid] < target) {
            cStart = mid +1,
        } else {
            cEnd = mid-1;
          }
    }
        return new int [ ] {-1, -1};
}
```