Binary Search

Algorithm :-

sorted array ↑

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

↑ ascending order

target = 36

1. Find the middle element
2. Check:
   if target > middle => search in right
   else => search in left
3. If target == middle => We found element

Example :-

In the above array

arr = [2, 4, 6, 9, (11), 12, 14, 20, 36, 48]
indexes: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

target: 36

1st -> Find the middle element

mid = $\frac{\text{start} + \text{end}}{2} = \frac{0+9}{2} = 4$ [The element at index 4 is the middle element]

2nd -> lets check:

Is target > middle => 36 > 11 => yes! => check right side

Now, arr = [2, 4, 6, 9, 11, 12, 14, (20), 36, 48]
indexes: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

mid = $\frac{5+9}{2} = 7$ [The element at index 7 is the middle element]

Lets check:

Is target > middle => 36 > 20 => yes! => Check right side

i.e, arr = [2, 4, 6, 9, 11, 12, 14, 20, (36), 48]
(8) (9)

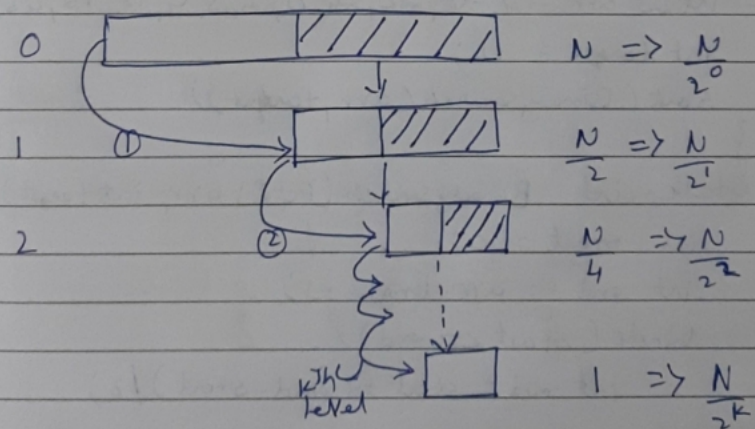
mid = $\frac{8+9}{2}$ = 8 [The element at index 8 is the middle element]

Here, target == middle => 36 == 36 => Element found at index 8

=> Time complexity:

Best case : O(1)
Worst case : O(log n)

Explanation : Find the maximum number of comparisons



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

$N = 2^k$

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

$\log n = 2 \log 2$

$k = \frac{\log N}{\log 2}$

$k = \log_2 N$ → size of array

→ Total no. of comparisions in worst case

=> Order agnostic Binary search

Let's search say if we don't know that the array is sorted in ascending (or) descending order

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

Here, target > middle => Search left

Here, start > end → descending order

When start < end → Ascending order

Code :- BS code

```
PSVm () {
 int[] arr = {-18, -12, -4, 0, 2, 3, 4, 15, 16, 45, 89};
 int target = 22;
 sout (Binarysearch (arr, target))
}
static int Binarysearch (int[] arr, int target) {
   int start = 0;
   int end = arr.length -1;
   while (start <= end) {
      int mid = start + (end - start) / 2;

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

Q: Ceiling of a number

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

Find the smallest element in array greater than (or) equal to target

Eg: ceil(arr, target = 14) = 14
ceil(arr, target = 15) = 16
ceil(arr, target = 4) = 5

Hint: Since, you are searching an element in an array of sorted elements you can apply Binary Search. if the target element is found in the array you can return the element else you can return array element of start index for next greater than the element target

Explanation:

Eg: arr = [2, 3, 5, 9, 14, 16, 18] target = 15

(0 1 2 3 4 5 6)

① -> [2, 3, 5, 9, 14, 16, 18] target > mid
s mid e

② -> [2, 3, 5, 9, 14, 16, 18] target < mid
s mid e

③ -> [2, 3, 5, 9, 14, 16, 18] target > mid
s, m, e

④ -> [2, 3, 5, 9, 14, 16, 18] -> Here the loop breaks since start > end
m s
e

Here, next greater than the target element will be number in start position

Code :-

```
PSVM () {
    int [] arr = {2, 4, 6, 9, 11, 12, 14, 20, 36, 45};
    int target = 13;
    Syout (ceilNumber (arr, target));
}

Static int ceilNumber (int [] arr, int target) {
    int start = 0;
    int end = arr.length - 1;

    if (target > arr [end]) {
        return arr [end];
    }

    While (start <= end) {
        int mid = start + (end - start) / 2;
        if (target > arr [mid]) {
            start = mid + 1;
        } else if (target < arr [mid]) {
            end = mid - 1;
        } else {
            return arr [mid]
        }
    }
    return arr [start];
}
```

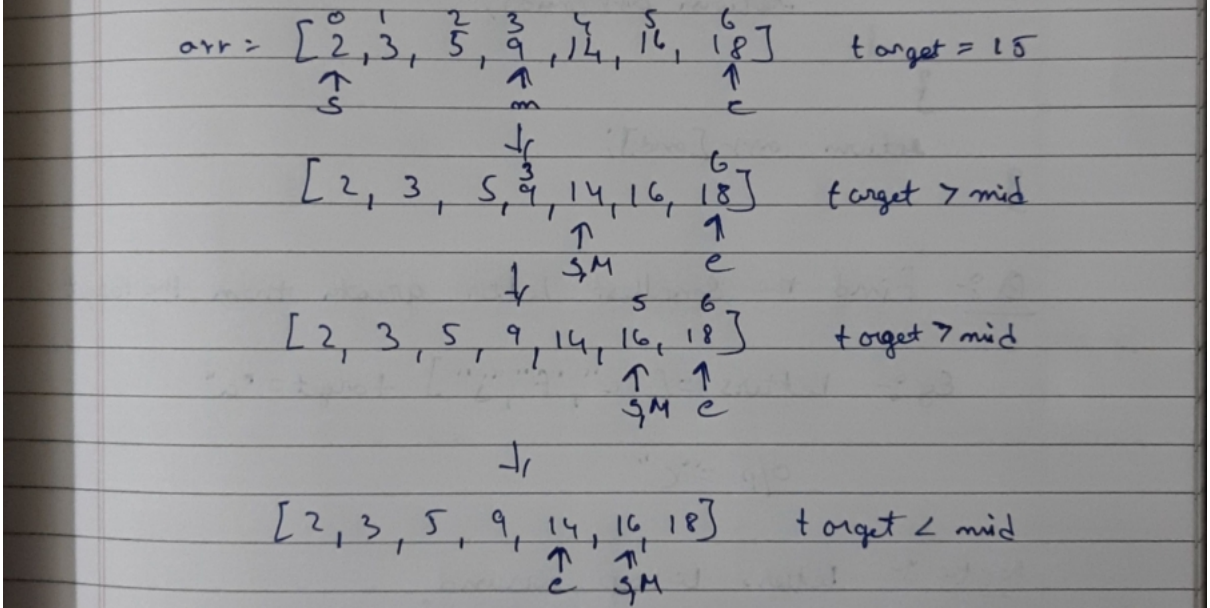
Q :- Floor of a number
arr = [2, 3, 5, 9, 14, 16, 18]; target = 14

Floor = Find the greatest element smaller or) equal to the target

Eg :- Floor (arr, 14) = 14
Floor (arr, 10) = 9
Floor (arr, 6) = 5

Hint :- Since, you are searching for an element in a sorted array you can apply Binary search. If the target element is found in the array you can return the element else you can return the array element of end index for smaller than the target element

Explanation :-



Here, the start > end loop breaks and return end index element for floor value

Code :-

```
PSVM(){
    int arr[] = {2,4,6,9,11,12,14,20,36,48};
    int target = 10;
    Sout(floor(arr, target))
}
Static int floor(int[] arr, int target){
    int start = 0;
    int end = arr.length - 1;
    if(target < arr[start]){
        return arr[start];
    }
```

```
While (start <= end){
    int mid = start + (end - start) / 2;

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

Q:- Find the Smallest letter greater than the target

Eg:- letters = ["c","f","j"] target = "a"

o/p = "c"

Note:- letters wrap around

Eg:- letters = ['a','b'] target = 'z'

o/p :- 'a' -> Should start from begi

Hint:- Same as ceiling of a number but return only greater than & remove equal sign

Code:-

```
PSVM (){
    char[] arr = {'c','f','j'}
    char target = 'j';
    Sout (nextChar(arr, target))
}
```

```
Static char nextCharacter(char[] arr, char target){
    int start = 0;
    int end = arr.length-1;

    While(start <= end){
        int mid = start + (end - start)/2;
        if(target > arr[mid]){
            start = mid+1;
        }else if(target < arr[mid]){
            end = mid-1;
        }else{
            start = mid+1;
        }
    }
    return arr[start % arr.length];
}
```

for wrap around

Q:- Find first and last position of element in sorted array

Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of given target value. if target is not found in the array return [-1, -1]

Eg:- nums : [5, 7, 7, 8, 8, 10] (indices 0 1 2 3 4 5), target = 8

o/p : [3, 4]

Hint:- Since, given array is a sorted array and target element to be searched we can use Binary Search. To find the first and last position. Find the index of first and last occurance and return the indexes

Code:-

```
Psvm(){
    int[] arr = {5,7,7,8,8,10};
```

```
    int target = 4;
    int[] ans = {-1, -1};
    ans[0] = searchIndex(arr, target, false);
    ans[1] = searchIndex(arr, target, true);
    Sout(Arrays.toString(ans));
}
static int searchIndex(int[] arr, int target, boolean isForward){
    int start = 0;
    int end = arr.length - 1;
    int ansIndex = -1;

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

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

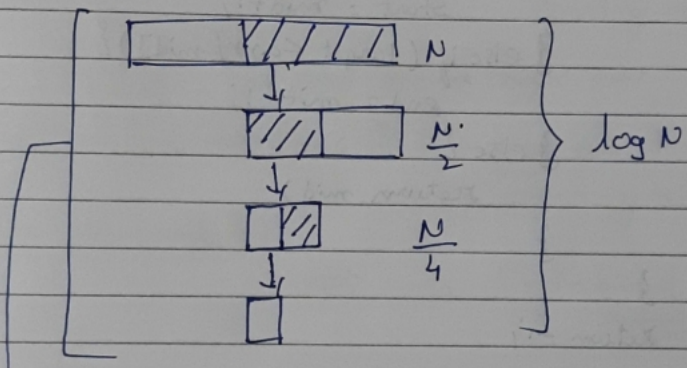
Q :- Find position of an element in a sorted array of infinite numbers

Suppose you have a sorted array of infinite array numbers, how would you search an element in the array

Hint :- Since, Searching is on sorted array we can use binary search. Here we dont have starting and ending bound so, we have to search the elements in chunks (like reversing of binary search)

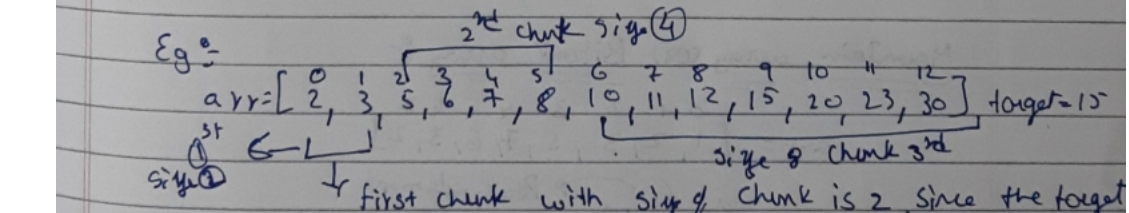
Explanation :-

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



→ Reverse this process to get chunks and increase the size of chunks. to

Eg :-



→ first chunk with size of chunk is 2, since the target is not in this chunk leave this chunk and create one more chunk with double the size of chunk

Code :-

```
PSVm () {
    int [] arr = {1, 2, 3, 5, 6, 7, 8, 10, 12, 13, 18, 20, 22};
    int target = 18;
    int start = 0
    int end = 1
    While ( target > arr[end]){
        int temp = end +1;
```

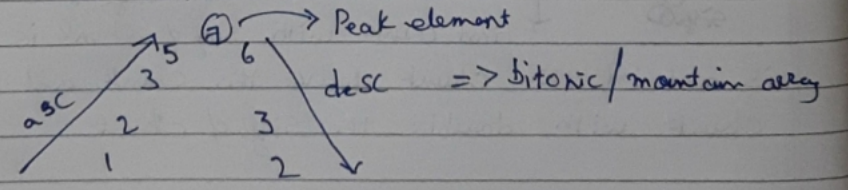
```
        end = end + (end - start + 1) * 2;
        start = temp;
    }
    sout (binary_search (arr, target, start, end));

static int binary_search (int[] arr, int target, int start, int end)
    while (start < end) {
        int mid = start + (end - start) / 2;
        if (target > arr[mid]) {
            start = mid + 1;
        } else if (target < arr[mid]) {
            end = mid - 1;
        } else {
            return mid;
        }
    }
    return -1;
}
```

Q :- Find the peak element in mountain array

Mountain array (or) Bitonic array :-

arr = [1, 2, 3, 5, 7, 6, 3, 2]
(indices 0 1 2 3 4 5 6 7)



Hint :- Since the given array is sorted array we can make use of binary search. here the main concept is to use binary search without knowing the target

Explanation :-

```
                S           M           E
                0   1   2   3   4   5   6   7
arr : [ 1 , 2 , 3 , 5 , 6 , 4 , 3 , 2 ]
        └ incrementing ┘   └ decreasing ┘
```

Cases :-

① If arr[mid] > arr[mid+1] => you are in the dec part of array

```
S   M   E
ans lies
```

=> e = mid
// you can stop looking right part

② arr[mid] < arr[mid+1] => you are in the asc part of the array

=> S = mid + 1

=> S M mid+1 E
ans lies in this part

③ When will loop break?

```
                0   1   2   3   4   5   6   7
arr : [ 1 , 2 , 3 , 5 , 6 , 4 , 3 , 2 ]
        S           M           E
```

//

```
                4   5   6   7
[ 6 , 4 , 3 , 2 ]
  S   M       E
```

//

```
                4   5
[ 6 , 4 ]
  S   E
  M
```

//

[6] => In the end S & E both will point to largest of number
S, E
Ans

Code :-

```
Psvm() {
    int[] arr = {1, 2, 3, 5, 6, 4, 3, 2};
    Sout(peak(arr));
}
Static int peak(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]){
            end = mid;
        }else{
            start = mid+1;
        }
    }
  Return start;
}
```

Q :- Search an element in rotated binary search

Rotated binary Search :-

Eg :-

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

1ST

After 1st rotation :- arr = [12, 2, 4, 5, 7, 8, 9, 10]

2nd

After 2nd rotation :- arr = [10, 12, 2, 4, 5, 7, 8, 9]

Hint :-

① Find peak element
② Binary search in asc array => (0, Pivot)
③ if not found, binary search in [Pivot, end]

Explanation :-

Eg :-

asc asc

arr : [3, 4, 5, 6, 7, 0, 1, 2]

↳ pivot => From where next numbers are asc

Case 1 :-

To find pivot

In the rotated sorted array when you find that mid > mid+1 i.e pivot

Eg :- [3, 4, 5, 6, 7, 0, 1, 2]

↳ only these 2 will be in dsc order

ans : mid

Case 2 :-

if mid element < mid-1 element

i.e also my ans => ans = m-1

Eg : [3, 4, 5, 6, 7, 0, 1, 2]

mid mid-1

0 < 7 ✓

Case 3 :-

Compare the mid with start element to decrease the range

Start element ≥ mid element

In this case all the elements from mid will be less than start

Smaller than start so we can ignore left side & right side

Eg :- [4, 5, 6, 3, 2, 1, 0]

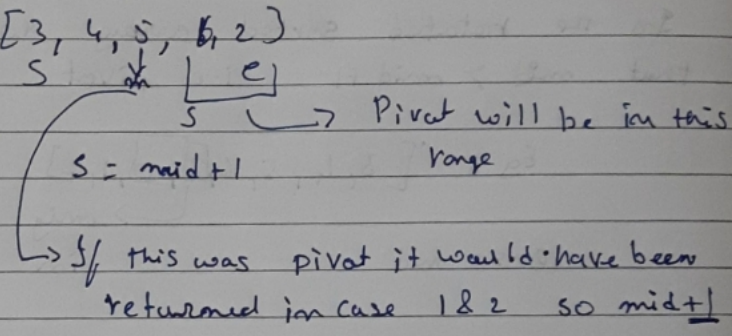
s mid e

Hence, we can ignore all these elements, since we are looking for peak i.e largest element

end = mid - 1

Case 4:- Compare with start element to decrease the range

Start element < mid element



Do binary search to search the target in the range

Case 1:- Pivot = target // ans

Case 2:- target > start element
Search space (start, Pivot-1)

case 3:- target < start element
i.e we know that all elements from start to pivot are going to be bigger then target

Search space (Pivot+1, till end)

Code:- ** For duplicate elements in array this algo doesn't work in this case if start element & end elements are equal you can skip those elements to find the pivot
Eg:- [2, 9, 2, 1, 2]
S M E

Code :-

```
psvm () {
    int arr = {1,2,3,4,5,6,7};
    sout(find_pivot_with_duplicates(arr));
}

Static int findpivot with duplicates (arr) {
    int start = 0;
    int End = arr.length - 1;

    while (start <= end) {
        int mid = start + (end - start) / 2;
        // 4 case over here
        if (mid < end && arr[mid] > arr[mid+1]){
            return mid;
        }
        if ( mid > Stat && arr[mid] < arr [mid-1]) {
            return mid-1
        }
        // if start == mid == end then skip duplicates
        if ( arr[mid] == arr[start] && arr[mid] == arr[end]){
            if ( start < end && arr[start] > arr[start+1]) {
                return start;
            }
        start ++;
        if ( end > start && arr[end] < arr [end-1]) {
            return end-1;
        }
        end ++;
    }
    else if ( arr [Start] < arr [mid] || arr[stat] == arr[mid]
            && arr [mid] > arr [end]){
        start = mid+1;
    } else {
        end = mid-1;
    }
} return -1; }
```

Q: Split array largest sum

Given an array nums which consists of non-negative integers and an integer m, you can split the array into m non-empty continuous subarrays

Write an algorithm to minimize the largest sum among these m subarrays

Explanation:-

arr = [7, 2, 5, 10, 8] , m = 2

Sub arrays

		largest
7, 2, 5, 10 (24)	| 8 (8)	24
[7, 2, 5] (14)	| [10, 8] (18)	18 Ans minimize the largest
[7, 2] (9)	| [5, 10, 8] (23)	23
[7]	| [2, 5, 10, 8] (25)	25

Cases:-

1) Minimum no. of partitions that we can make : 1

[7, 2, 5, 10, 8] if m = 1

32

2) Max number of partitions is N because subarrays cannot be empty

[7], [2], [5], [10], [8] if m = N

[10]

*** So, sum of sub arrays lie in the range

[10, 32] -> The min possible sum of the array is in this range

*** So, here we can use Binary search in this range

Break down:-

Range = [10, 32]

start = 10 ; end = 32 => mid = $\frac{s+e}{2} = \frac{42}{2} = 21$

Try to see if we can split the array with 21 as the max sum

[7, 2, 5, 8, 10] Pieces = 2

[7, 2, 5] [8, 10]

14 18

Check 1: if (pieces $\leq$ m) => end = mid

Now: start = 10 ; end = 21 ; mid = 15

[7, 2, 3], [8], [10] Pieces = 3

Check 2: if (Pieces > m) => start = mid + 1

Now:- start = 16; end = 21 ; mid = 18

[7,2,5] , [8,10] pieces = 2

Now:- start = 16 ; end = 18 ; mid = 17

[7,2,5] , [8], [17] pieces = 3

s = m+1 = 18

s = 18 ; e = 18

m = 18 ;

Ans

// The ans exists definetly, here by the above 2 checks we will reach

Code :-

```
Public int SplitArray(nums, int m){
    int start = 0;
    int end = 0;

    for(int i=0; i<nums.length; i++){
        start = Math.Max(start, nums[i])
            // in the end of the loop this will chontain
                the max item of the array
        end += nums[i];
    }
    While(start < mid){
        int mid = start + (end-start)/2;
        int sum = 0; int pieces = 0;
        for(int num : nums){
            If(sum + num > mid){
                // you cannot add this in this array so
                    new one
                sum = num;
                Pieces ++
            }
```

```
        else {
            sum += num;
        }
        if(pieces > m) {
            start = mid+1;
        }else{
            end = mid;
        }
    }
    return end; // here start = end;
}
```

Binary Search in 2D array

Searching in matrices :-

$$\begin{bmatrix} 18 & 9 & 12 \\ 36 & -4 & 91 \\ 44 & 33 & 16 \end{bmatrix}$$ Target = 91

We can do linear search for this matrix since, the ~~matrix~~ matrix is not sorted in any manner

Code :-

```
Static int[] search(arr, target){
    for(r=0; r<n; r++){
        for(c=0; c<n; c++){
            if(arr[r][c] == target){
                return new ~~int[]~~ {r, c};
            }
        }
    }
    return -1;
}
```

$O(N^2)$, $O(1)$

Q :- Matrix is sorted in a row-wise & column-wise manner.

Eg :-

	0	1	2	3
0	10	20	30	40
1	15	25	35	45
2	28	29	37	49
3	33	34	38	50

(asc)

Target = 37

Hint :- Since the rows and columns are sorted we can make use of binary search and decrease the search space

Explanation :- Start searching from row = 0 ; col = last col

Case 1 :- If element == target
=> ans found

Case 2 :- If element < target
=> row ++

Case 3 :- If element > target
=> Column --

LB			
10	20	30	40
15	25	35	45
28	29	37	49
33	34	38	50

=> upper bound

Compare search element with the up & lb and eliminate rows & cols accordingly

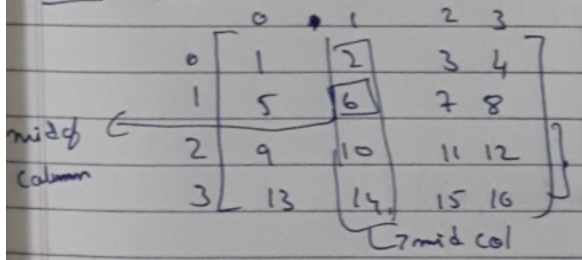
Code :-

```
PSVm () {
    int[][] arr = {
            {10, 20, 30, 40},
            {15, 25, 35, 45},
            {28, 29, 37, 49},
            {33, 34, 38, 50}}; sout(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 which is strictly sorted



* Take middle col & perform BS on it

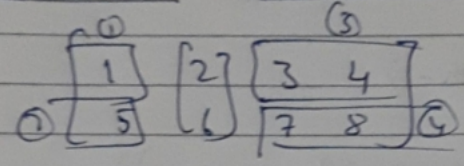
* If mid of the colum == target ans found
if m of colm < target {
}

** Start search element from in mid of row & mid of col

Cases :-

Case 1 :- if element == target => ans found
Case 2 :- if element > target => // Ignore rows after it
Case 3 :- if element < target => // ignore rows above it

In the end 2 rows are remaining :-



1) Check whether the mid column you are at that contains the answer [2, 6]
was found in this case

in case target = 3 ?

(2) Consider the 4 parts and perform bs on these parts

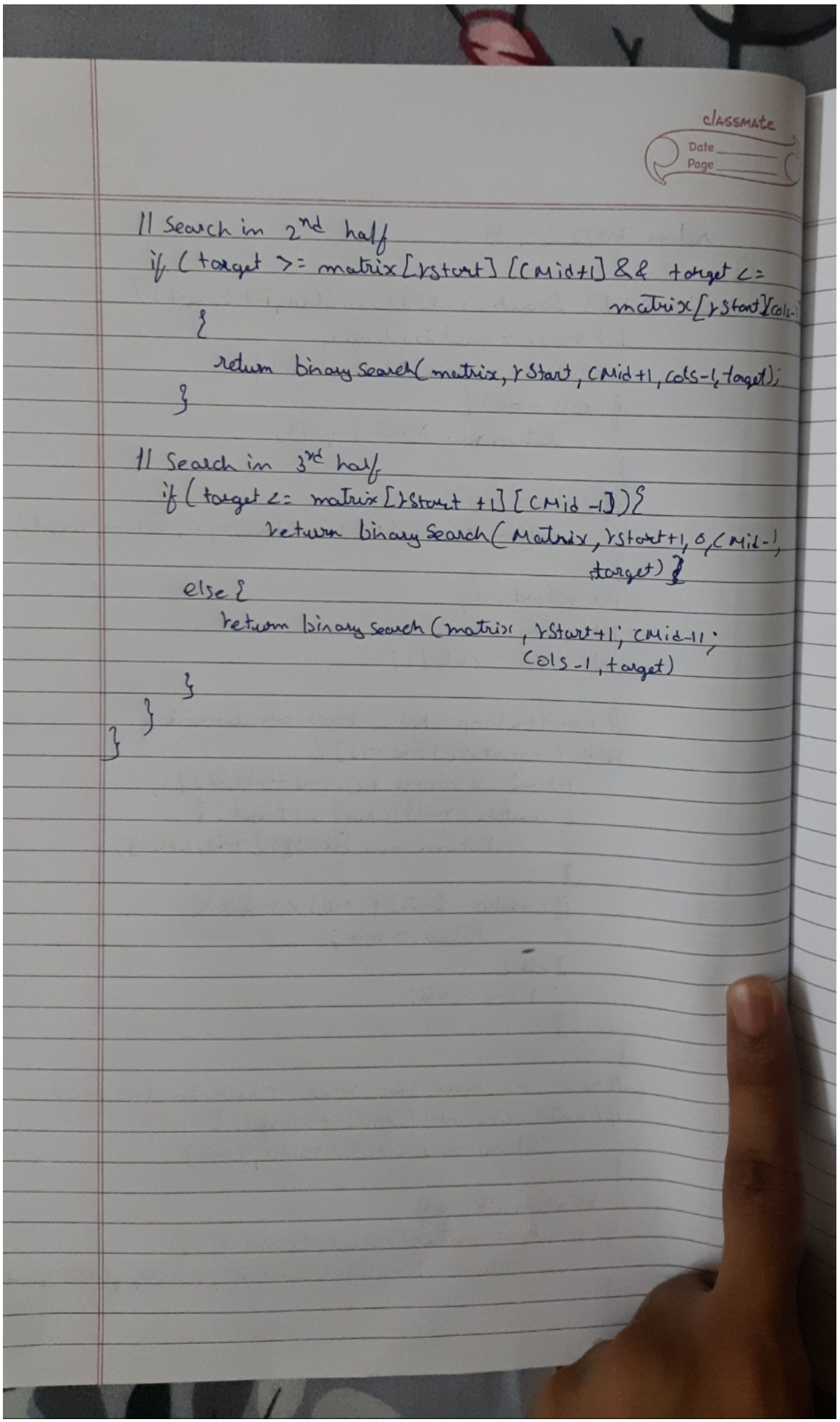
Code :-

```
PSvm () {
    int[][] = {
        {1, 2, 3,},
        {4, 5, 6},
        {7, 8, 9}
    }
    Sout( Search (arr, 9))
}

// Search in rows provided between cols provided
static int[] binary search(int[][] matrix, int cstart, int cEnd,
                           int 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 int[] {-1, -1}
}
Static int [] Search (int[][] matrix, int target) {
        int rows = matrix.length;
        int cols = matrix[0].length - 1;
        if(cols == 0) {
            return new int[] {-1, -1};
        }
        if (rows = 1){
            return binarySearch(matrix, 0, 0, cols-1, target);
        }
        int rStart = 0;
        int rEnd = rows-1;
        int cMid = cols/2;

        // run the loop till 2 rows are remaining
        While (rStart < (rEnd -1)) {
            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 two rows check in two rows
        if(matrix [rStart][cMid] == Target) {
            return new int[] { rStart, cMid}
        }
        // search in 4 half
        if(target <= matrix[rStart][cMid-1]) {
            return binarySearch(matrix, rStart, 0, cMid-1, target)
        }
```



```
// Search in 2nd half
if (target >= matrix[rStart][cMid+1] && target <=
                                    matrix[rStart][cols-1])
    {
      return binarySearch(matrix, rStart, cMid+1, cols-1, target);
    }

// Search in 3rd half
if (target <= matrix[rStart+1][cMid-1]){
        return binarySearch(Matrix, rStart+1, 0, cMid-1,
                                                target)}
  else{
      return binarySearch(matrix, rStart+1, cMid-1,
                                    cols-1, target)
    }
  }
}
```

PDF Created Using

Camera Scanner

Easily Scan documents & Generate PDF

https://play.google.com/store/apps/details?id=photo.pdf.maker