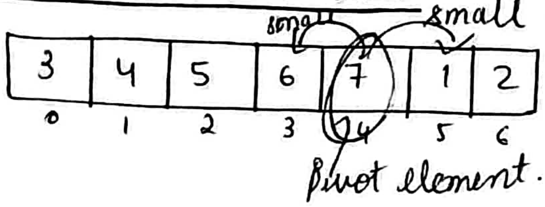
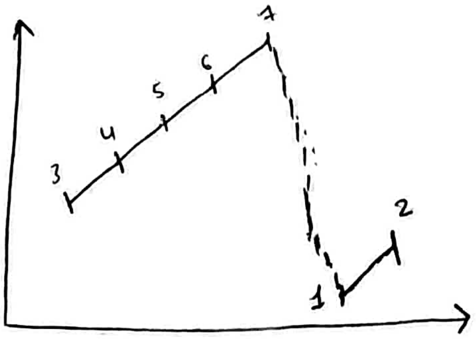
Lec - 11 18 Feb

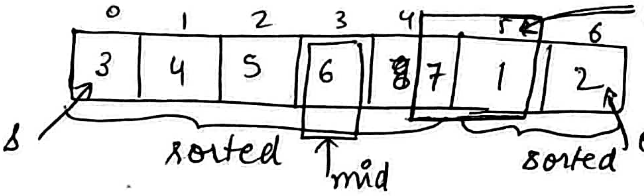
Searching and sorting -2

Q. Find Pivot Element →





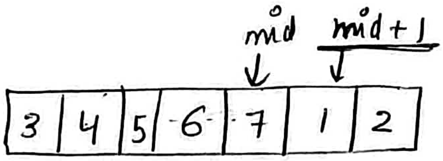
Brute force → Maximum element in the array. O(n) T.C.



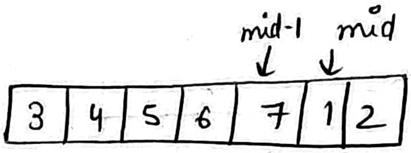
We can stuck in only this part, no matter what the length (size) of the array. So we will handle these 2 case seperatly.

s = 0
e = 6
mid = $\frac{0+6}{2}$ = 3

if (arr[mid] > arr[mid+1])
return mid; →

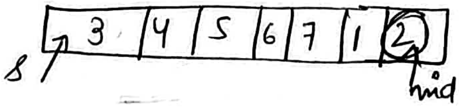


if (arr[mid-1] > arr [mid])
return mid-1; →



We are left with 2 conditions Now (search in left or search in right).

if (arr[s] > arr[mid]
search in left part. →



3>2 → search in left.

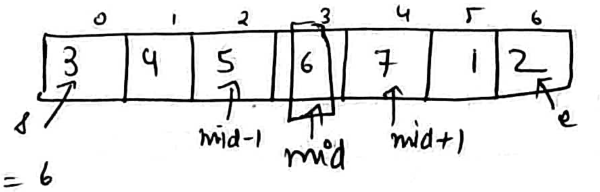
else if (arr[s] < arr[mid]
search in right part.

Code -

```
vector<int> arr{3, 4, 5, 6, 7, 1, 2};
int n = arr.size()-1;
int s = 0, e = n-1;
~~while (~~
int mid = s+ (e-s)/2;
while ( s<=e ){
        if ( mid+1 < arr.size()&& arr[mid] >arr [mid+1])
                return mid;
```

```
    if (mid-1> =0 && arr[mid-1] > arr[mid])
            return mid-1;
    if (arr[s] >= arr[mid])
            e = mid-1;
    else
            s = mid+1;
    mid = s + (e-s)/2;
}
return -1;
```

dry run-



s = 0 , e = 6

mid = $\frac{0+6}{2}$ = 3

arr[mid] = 6 , arr[mid-1] = 5 , arr[mid+1] = 7

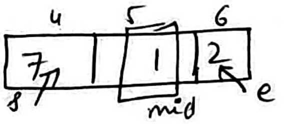
6 > 7 → F

5 > 6 → F

arr[s] >= arr[mid]

3 >= 6 → F

s = mid + 1



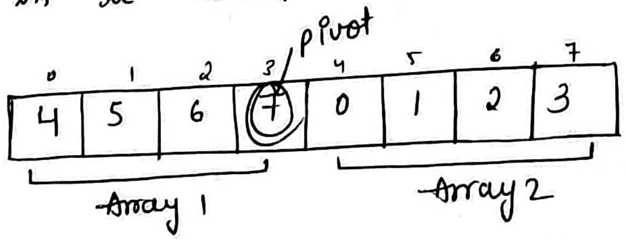
mid = 5

1>2 → F

7>1 → T ↳ return 7 as ans.

Q → Search in a Sorted and Rotated Array →

leetcode (33)



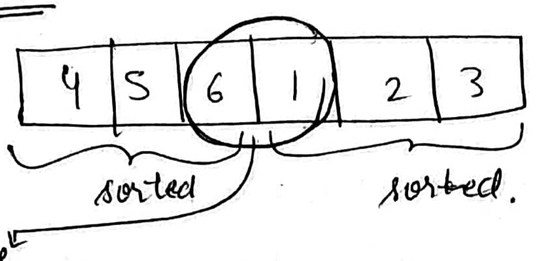
, target = 2
0 to n-1

4 to 7   0 to 3

⇓

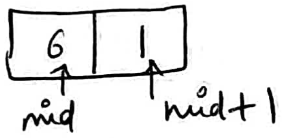
2 can't exist between these so we will apply binary search in Array 2.

let's start again

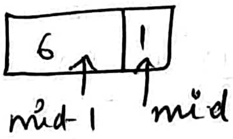


→ **find pivot**

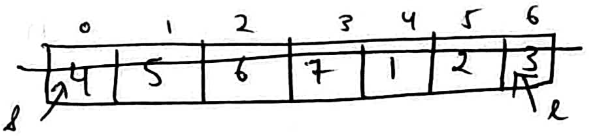
Two case

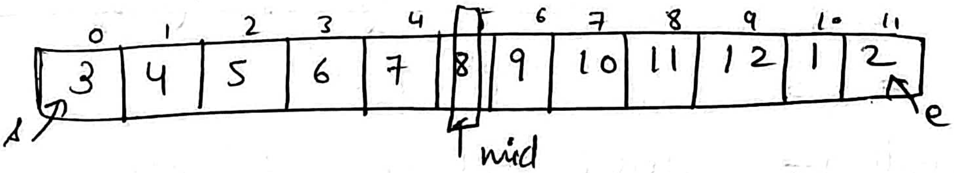


if (arr[mid] > arr[mid+1]
return mid;

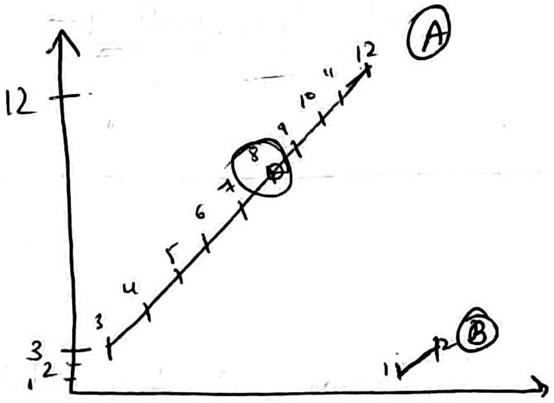


if (arr[mid-1] > arr[mid])
return mid;





Observation → All the elements of B will be less than line A, so pivot can't be in line B.



⇒ if (arr[mid] ~~=~~> arr [s])
↳ go to right
s = mid+1;

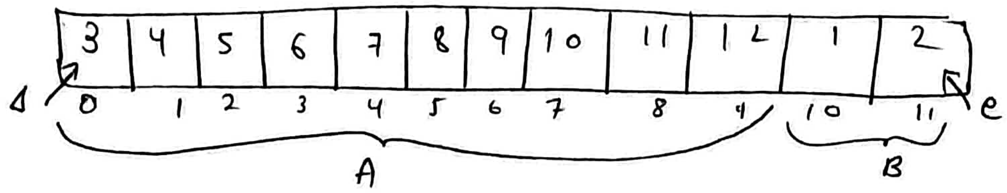
mid in line A.

else go to left
e = mid -1;

mid in line B

⇒ If we have only one element that element will be pivot. We don't need to run while loop.
We will simply return the element.
That's why while (s < e) condition.

now search →



, target = 2

A → 0 → pivot

B → pivot + 1 → n-1

Brute force – Search (Binary Search) in both the arrays. If the element is really present than you will find that element.

$T.C = O(\log n) + O(\log n)$

$= O(2\log n) = O(\log n)$

Better way is to search in the correct array.

↳ A → 03 → 12 → target = 2 can't exist here

B → 1 to 2 → target = 2 can exist here so search in this array.

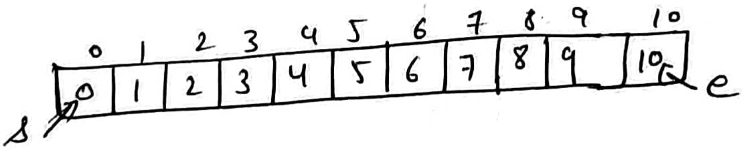
⊙ Square Root of a number using Binary Search →

i/p $\sqrt{12} = \boxed{3}$ only integer part

Observation – square root of 12 can lie between 0 to 12.

↳ so this can be our search space. so we will apply binary search.

$\sqrt{10}$ → 0 to 10



s = 0, e = 10

$mid = \frac{0+10}{2} = 5$

mid * mid > target
↳ search in left.

mid * mid < target
↳ store ans
↳ search in right.

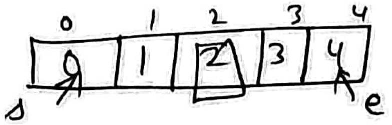
if (mid×mid == n)
↳ ans = mid;
return ans;

$5 \times 5 = 25$

$25 < 16 \rightarrow F$

$25 > 16 \rightarrow T$

search in left



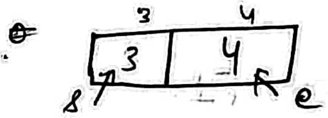
$\text{mid} = \frac{0+4}{2} = 2$

$2 \times 2 = 4$

$4 < 16 \rightarrow T$

↳ store ans ans = $\boxed{2}$

↳ search in ~~left~~. right



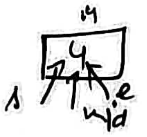
$\text{mid} = \frac{3+4}{2} = 3$

$3 \times 3 = 9$

$9 < 10 \rightarrow T$

↳ store ans $\boxed{3}$

↳ search in right



mid = 4

$4 \times 4 = 16$

$16 < 10 \rightarrow F$

$16 > 10 \rightarrow T$

↳ search in left.

s = 4, e = 3

↳ s>e → stop.

(6) n=25

s=0   e=25

$\text{mid} = \frac{0+25}{2} = 12$

$12 \times 12 > 25 \rightarrow F$

↳ left

s = 0 , e = 11

mid = 5

$5 \times 5 == 25 \longrightarrow T$

return ans = 5

```
~~vector<int> arr{~~
~~int s=0; e= size arr.size()-1;~~
main(){
    int n;
    cin>>n;
    int s=0, e= n ;
    int mid = s+ (e-s)/2;
    int ans=0 ;
    while (s<=e){
        if ( mid* mid == target)
                return mid;
        else if (mid * mid < target){
                ans = mid;
                s = mid+1;
        }
        else
                e= mid -1;
        mid = s +(e-s)/2;
    }
    return ans;
}
```

Now floating part →

$\sqrt{10} = 3.16$

we know ans =3.

3.1 → 3.1 * 3.1 <=10 → T → store → 3.11 → 3.11 * 3.11 < 10 → T

3.2 → 3.2 * 3.2 < 10 → F stop

3.3

3.12 → 3.12 * 3.12 < 10 → T

3.13 → 3.13 * 3.13 < 10 → T

3.14 → 3.14 * 3.14 < 10 → T

3.15

3.16

3.17 → 3.17 * 3.17 < 10 → F stop

This is just brute force not binary search.

Code →

```
int precision;
cout << "Enter the number of floating digits" << endl;
cin>> precision;
double step =0.1;
double finalAns = ans;
```

```
for (int i=0; i< precision; i++){  ← loop for no. of
                                     floating point.
    for ( double j= finalAns; j*j<=n; ~~j++)~~ j=j+step){
        finalAns = j;
    }
    step = step /10;
}
cout << finalAns ;
```

dry run-

precision = 2

finalAns = 3 step ~~0.1~~ 0.01 precision 2

step = 0.1

i = 0 , 0 < 2 → T

j = 3

3 * 3 = 9 < 10 → T

j = 3 + 0.1 = 3.1

3.1 x 3.1 = 9.61 <=10 T

j = 3.1 + 0.1 = 3.2

3.2 x 3.2 = 10.24 ~~→ F~~ <=10 → F

step = 0.1/10 = 0.01

i = 1 , i < precision , 1 < 2 → T

j = 3.1

3.1 x 3.1 = 9.61 <=10 → T

j = 3.1 + 0.01 = 3.11

3.11 X 3.11 = 9.6721 <=10 → T

j = 3.11 + 0.01 = 3.12

3.12 x 3.12 = 9.7344 <=10 → T

| |

j = ~~3.16~~ 3.15 + 0.01 = 3.16

3.16 * 3.16 = <=10 → T

j = 3.16 + 0.01 = 3.17

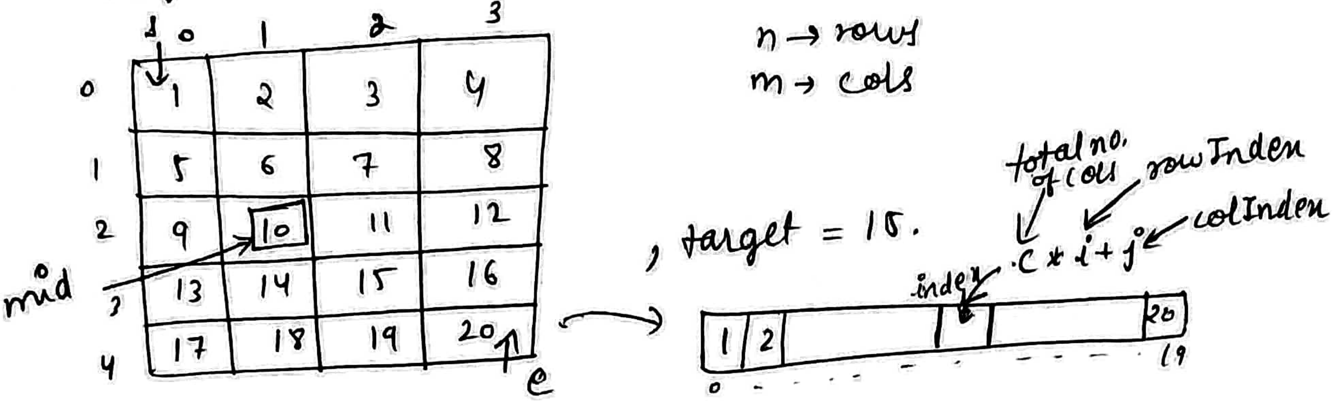
3.17 * 3.17 = <= 10 → F

i = 2 , i < precision → False

↳ return finalAns.

Find Out can we do this by binary search.

⊙ Binary Search in 2D matrix →



	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16
4	17	18	19	20

, target = 15.

$s = 0$, $e = n * m - 1 = 20 - 1 = 19$

$mid = \frac{0+19}{2} = 9$

How can we find index of row and col by mid?

row Index = mid / cols. = 9 / 4 = 2

col Index = mid % cols = 9 % 4 = 1

mid ⇒ $\boxed{arr[2][1] = 10}$

if (15 == 10) → found
if (15 < 10) → left
if (15 > 10)
    ↳ right

if (arr [row Index][colIndex] < ~~15~~ target) → right.

Algo / Pseudo code →

```
int arr[5][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16},{17,18,19,20}};
int n = 5, ~~cat~~ m = 4;
int s = 0;          rows  cols
int totalSize = n * m ;
int e = totalSize -1;
int mid = s + (e-s)/2;
while (s <= e){
    int rowIndex = mid/cols; = mid/m;
    int ~~rowo~~ colIndex = mid % m;
    if (arr[rowIndex][colIndex] == target)
        cout << "found" << endl;

    else if (arr[rowIndex][colIndex] < target)
        s = mid + 1;

    else
        e = mid-1;
    mid = s+(e-s)/2;
}
```