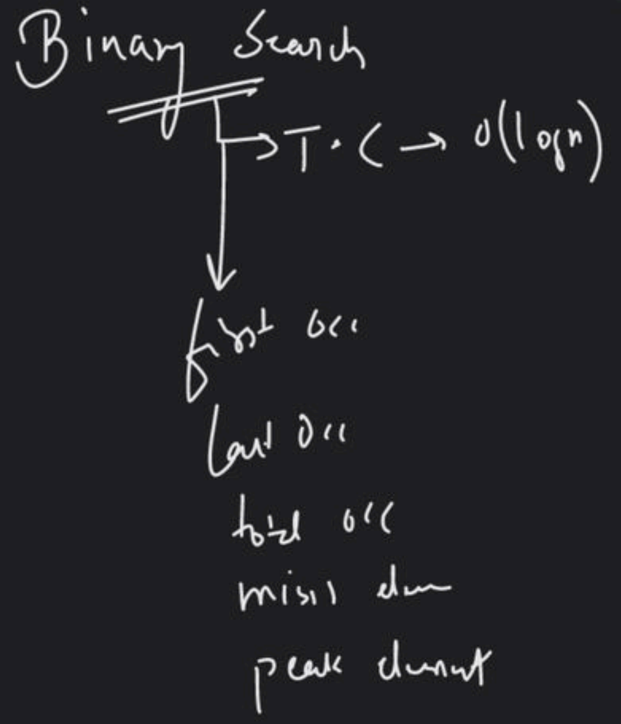
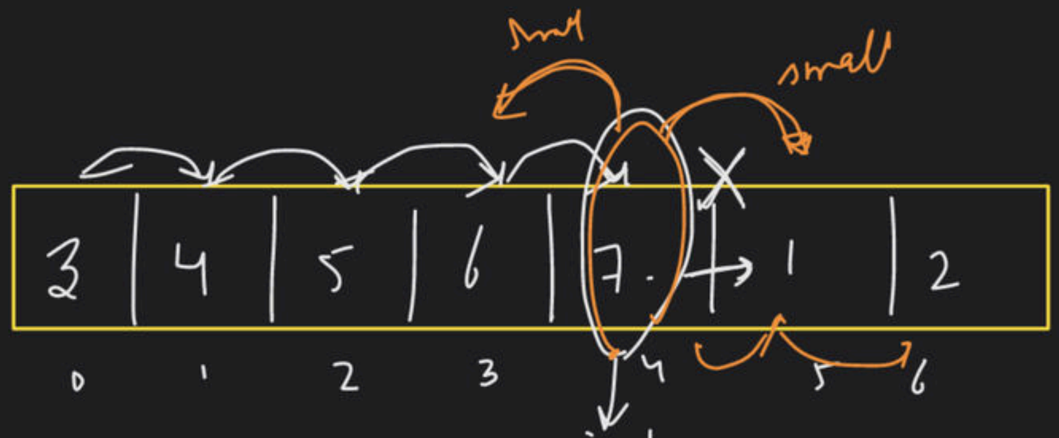
Searching and Sorting Class - II

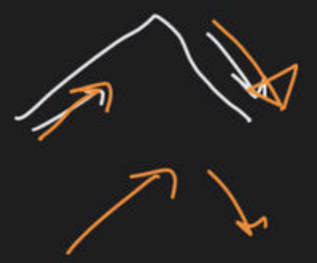
Special class



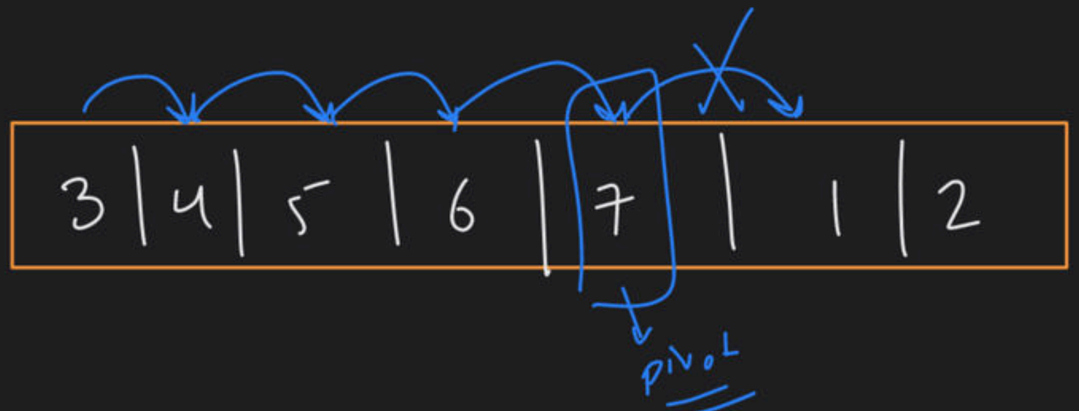
Love Babbar • Feb 18, 2023

→ find pivot element





what is pivot?

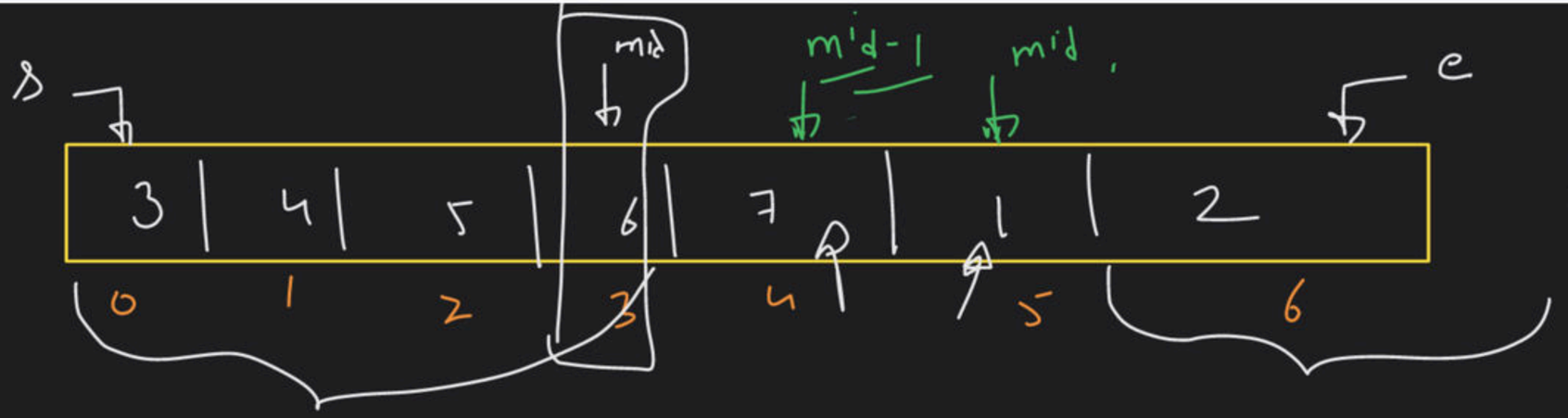


Brute Force → Linear Search → T.C → O(n)

s = 0

e = 6

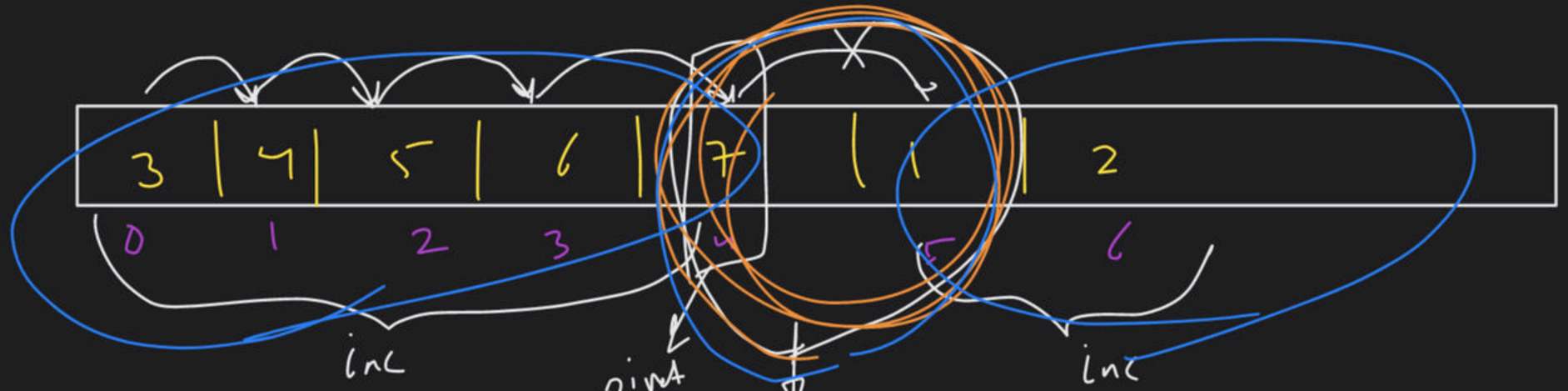
mid = $\frac{0+6}{2}$ = 3



pivot = 7

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

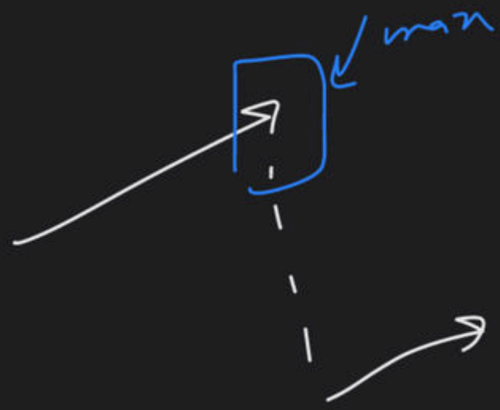
```
if (arr[mid-1] > arr[mid])
    return mid-1;
```



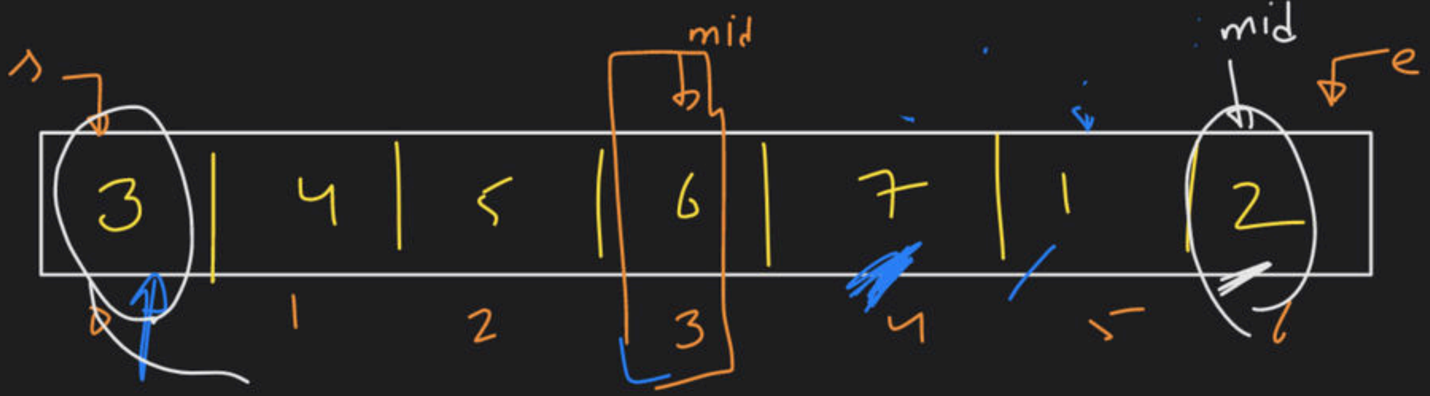
pivot
7

find the pivot element

handle edge case



s = 0
e = 6
mid = (0+6)/2 = 3



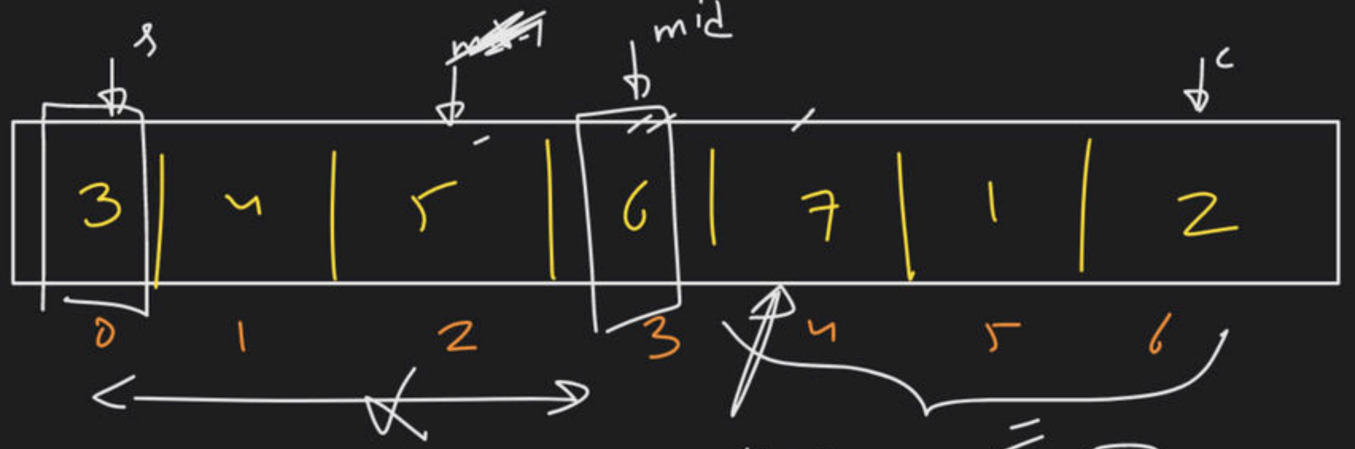

```
if (arr[s] > arr(mid))  // left
    {
        e = mid - 1
    }

else if (arr[s] < arr(mid))  // right
    {
        s = mid + 1
    }
```

```
if (arr[mid] > arr[mid+1])
    return mid

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

return -1
```

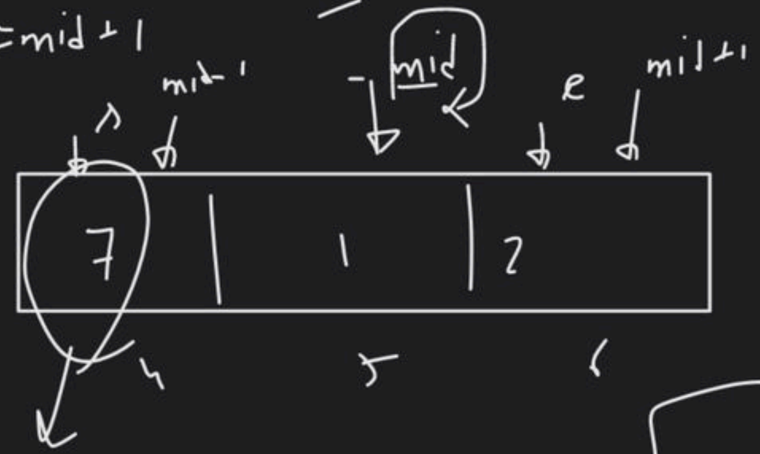
$s = 0$

$e = 6$

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



$s = mid + 1$

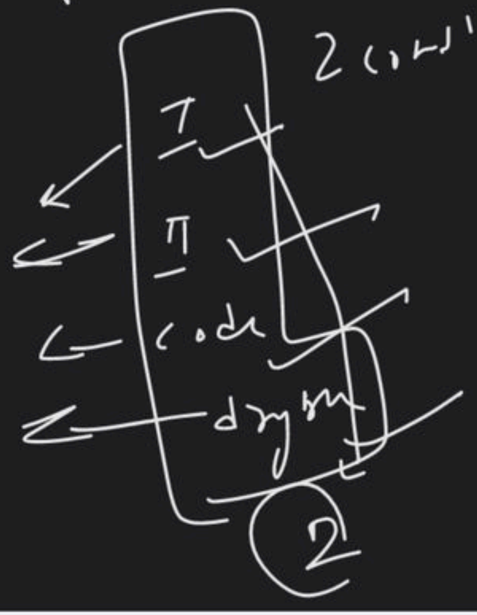


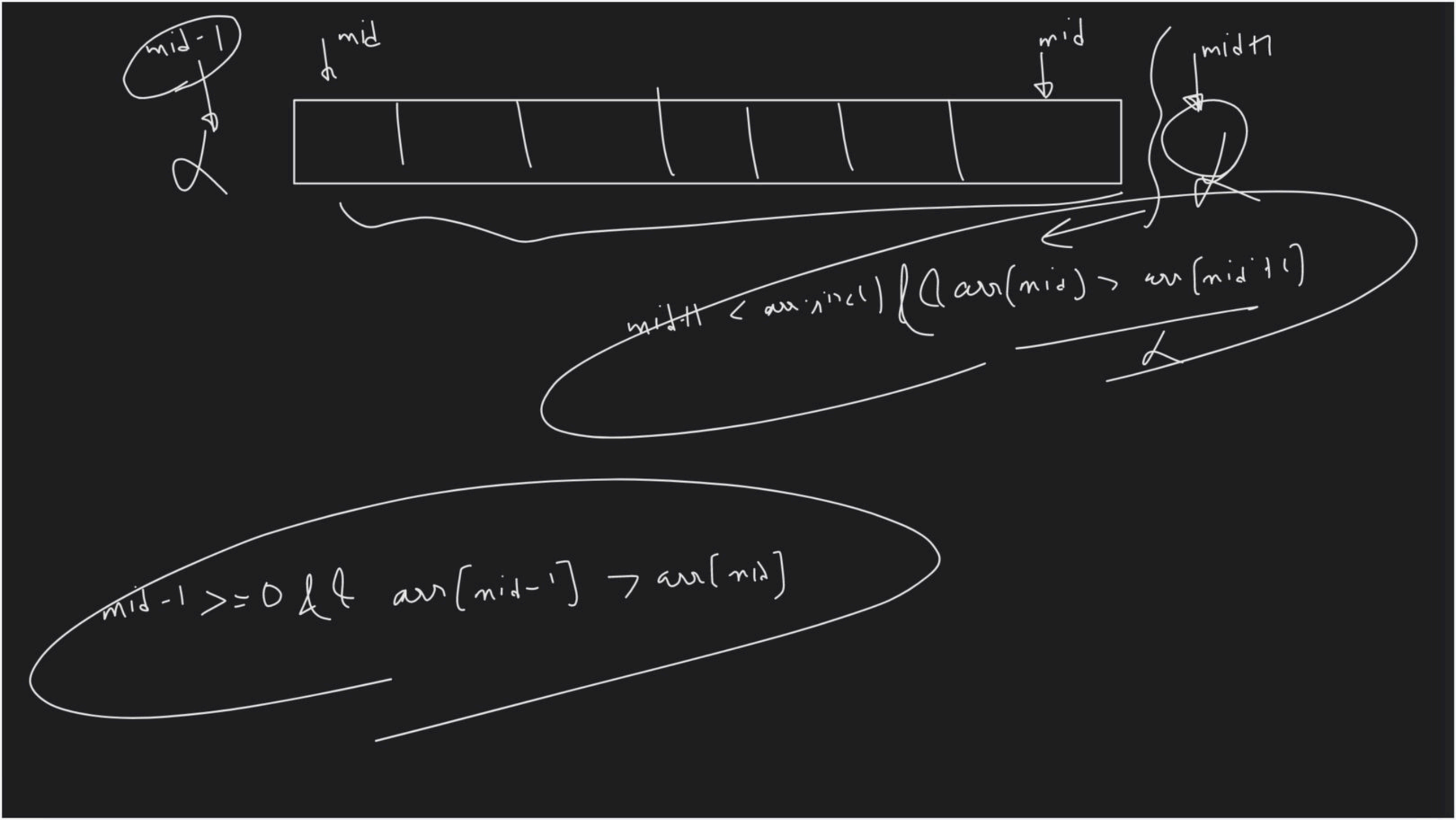
$s = 4$

$e = 6$

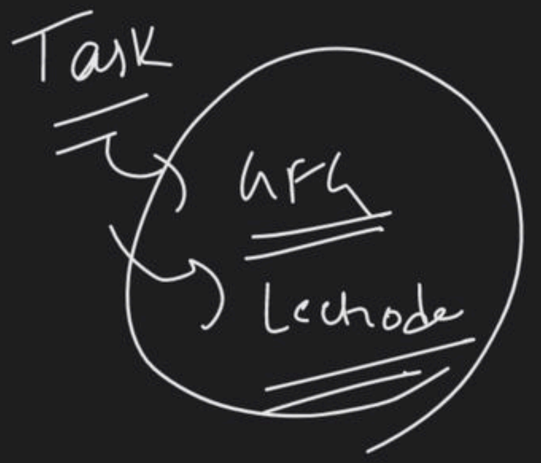
$mid = \frac{4+6}{2} = 5$

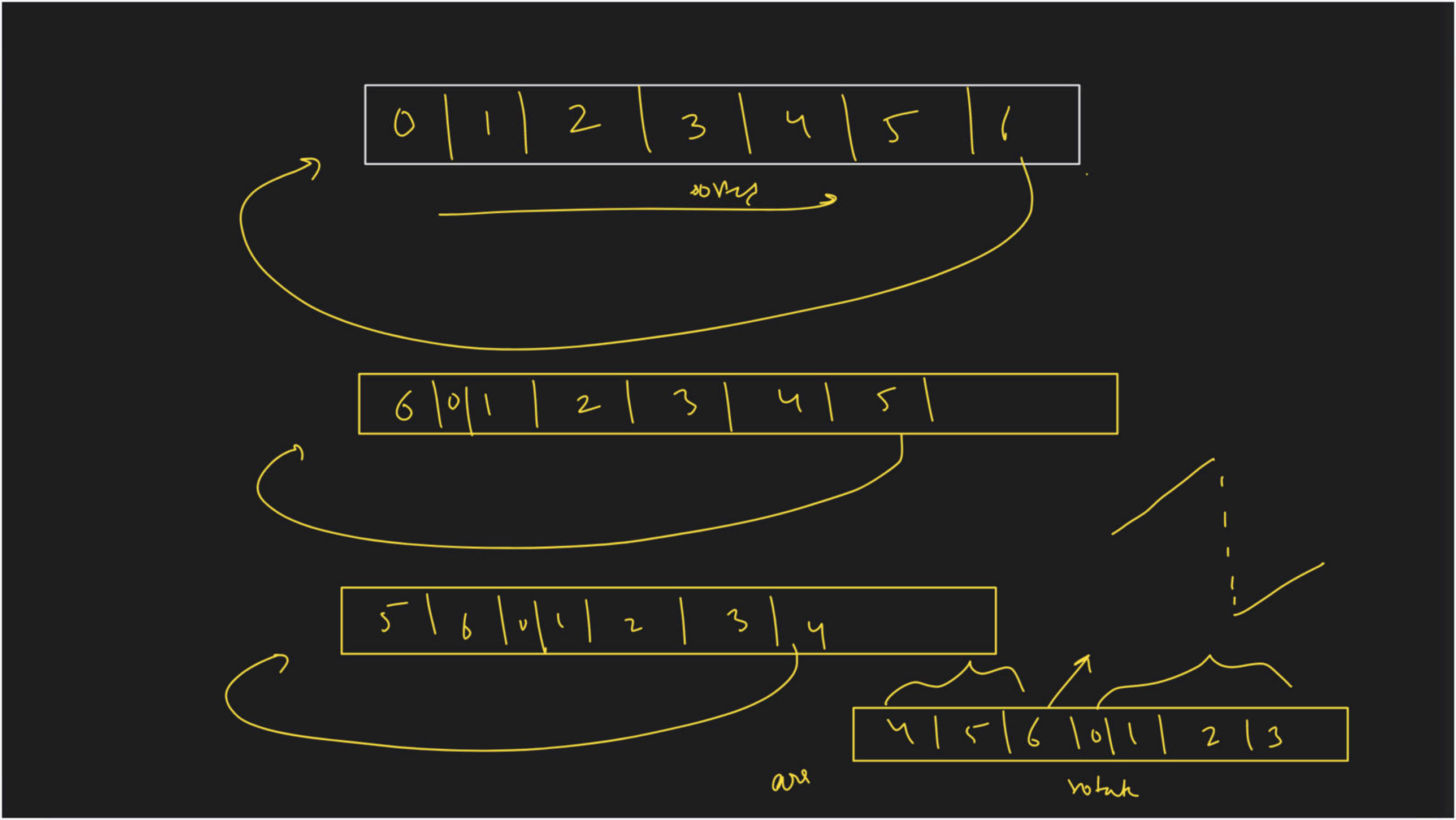
return

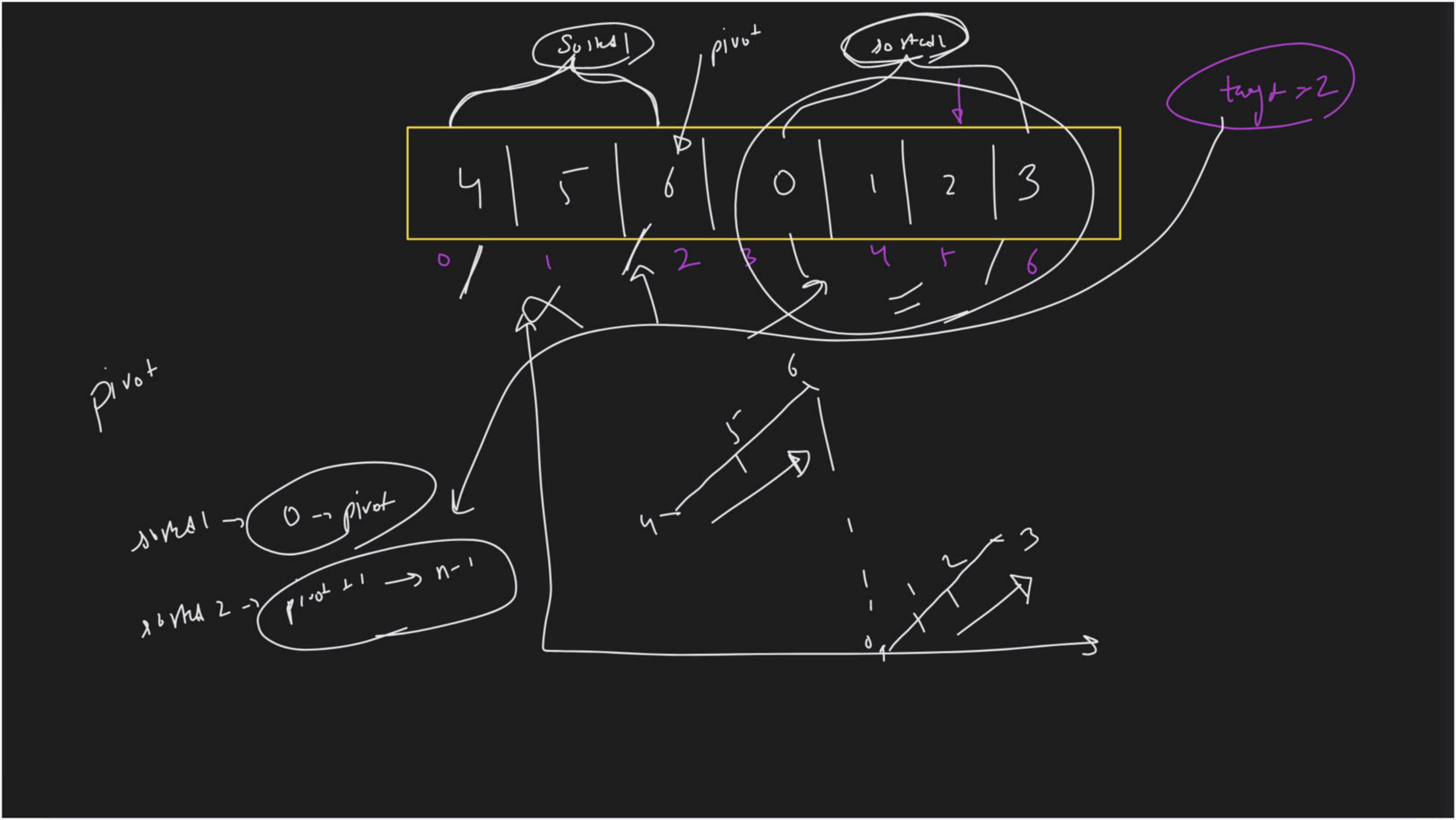


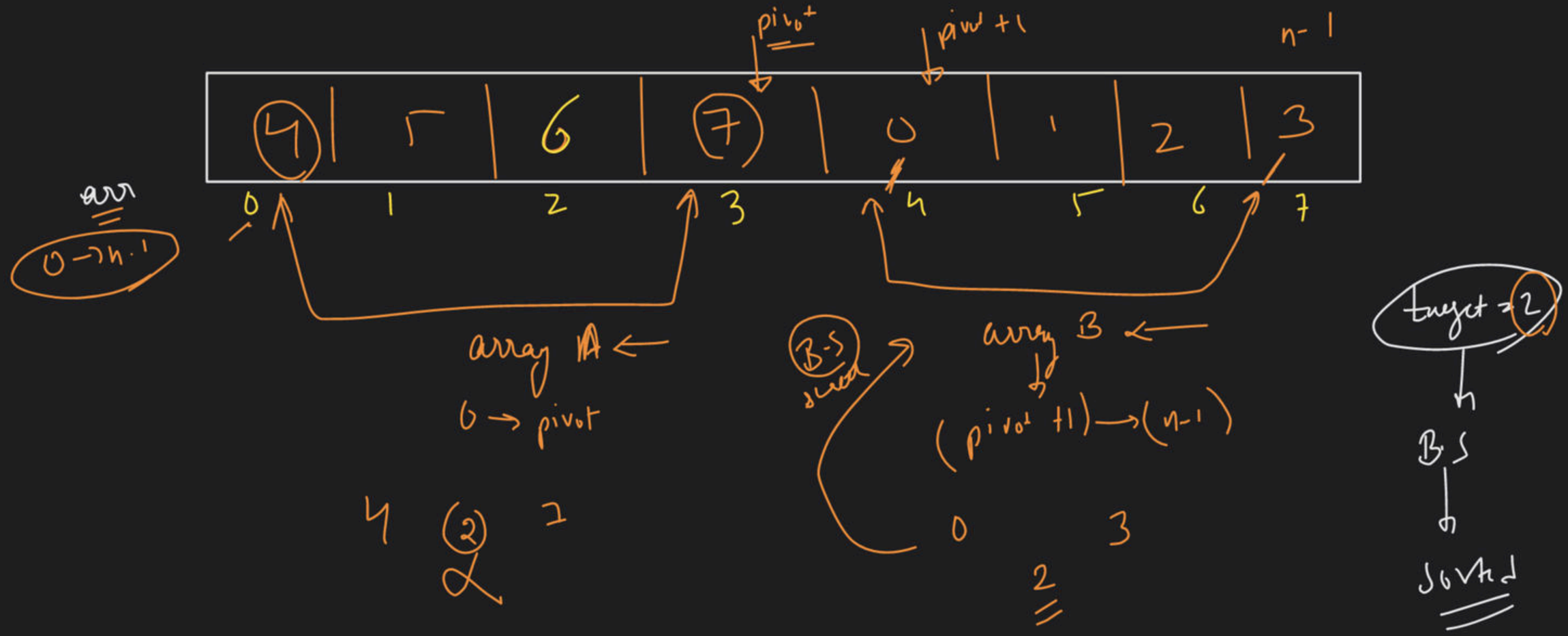

mid-1
mid
mid
mid+1
mid+1 < arr.size() && arr[mid] > arr[mid+1]
mid-1 >= 0 && arr[mid-1] > arr[mid]

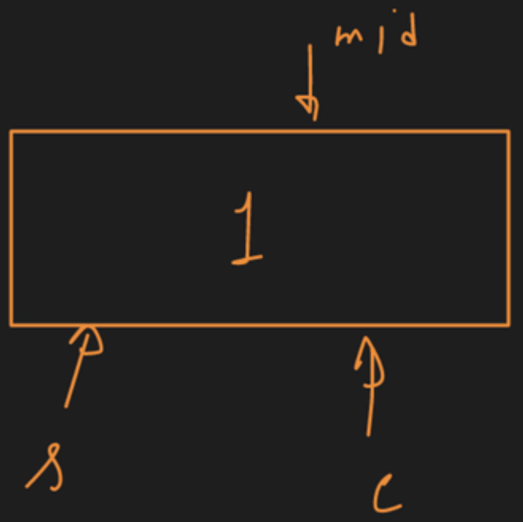

1min

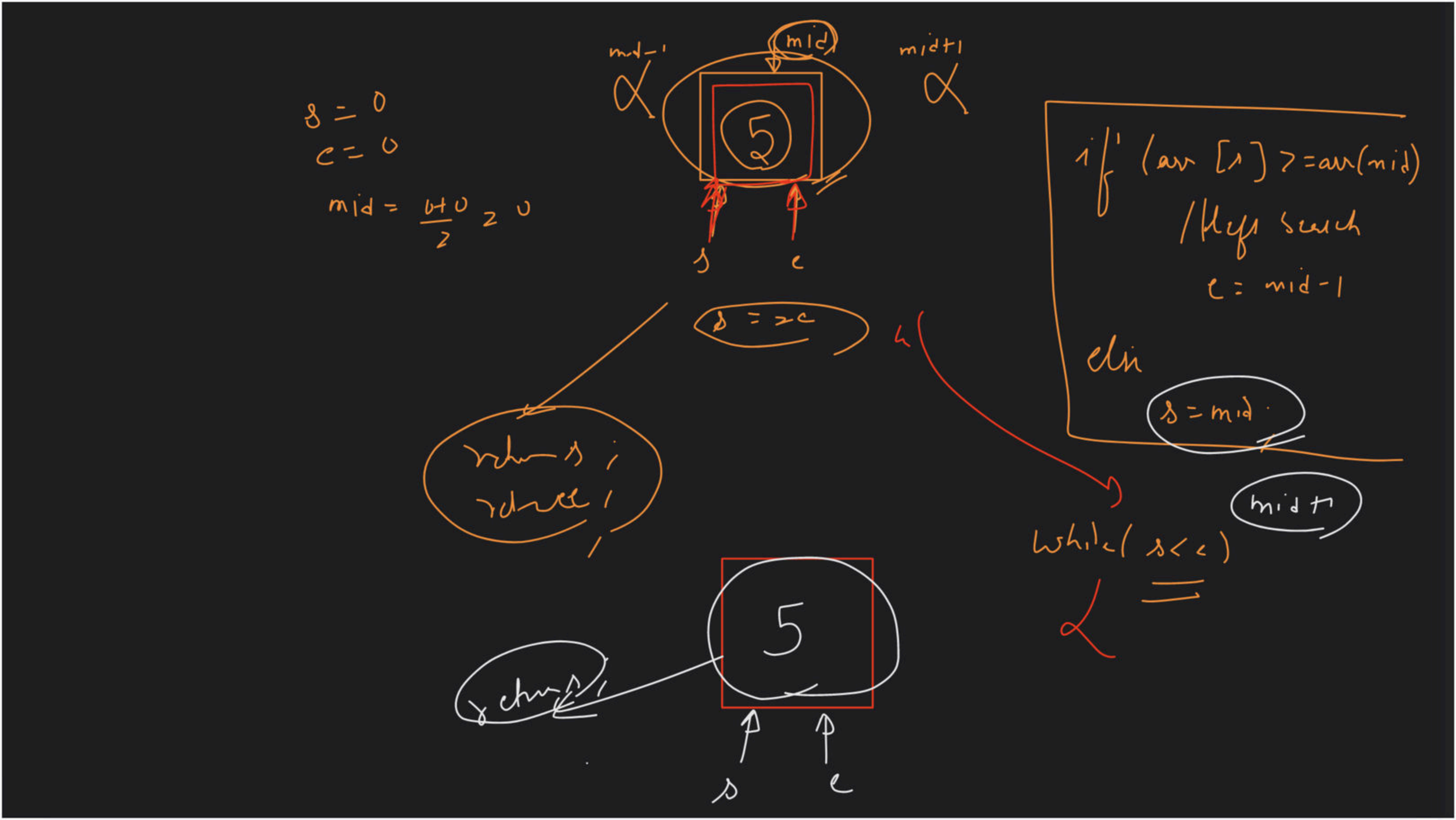

Task
GPG
Lecture

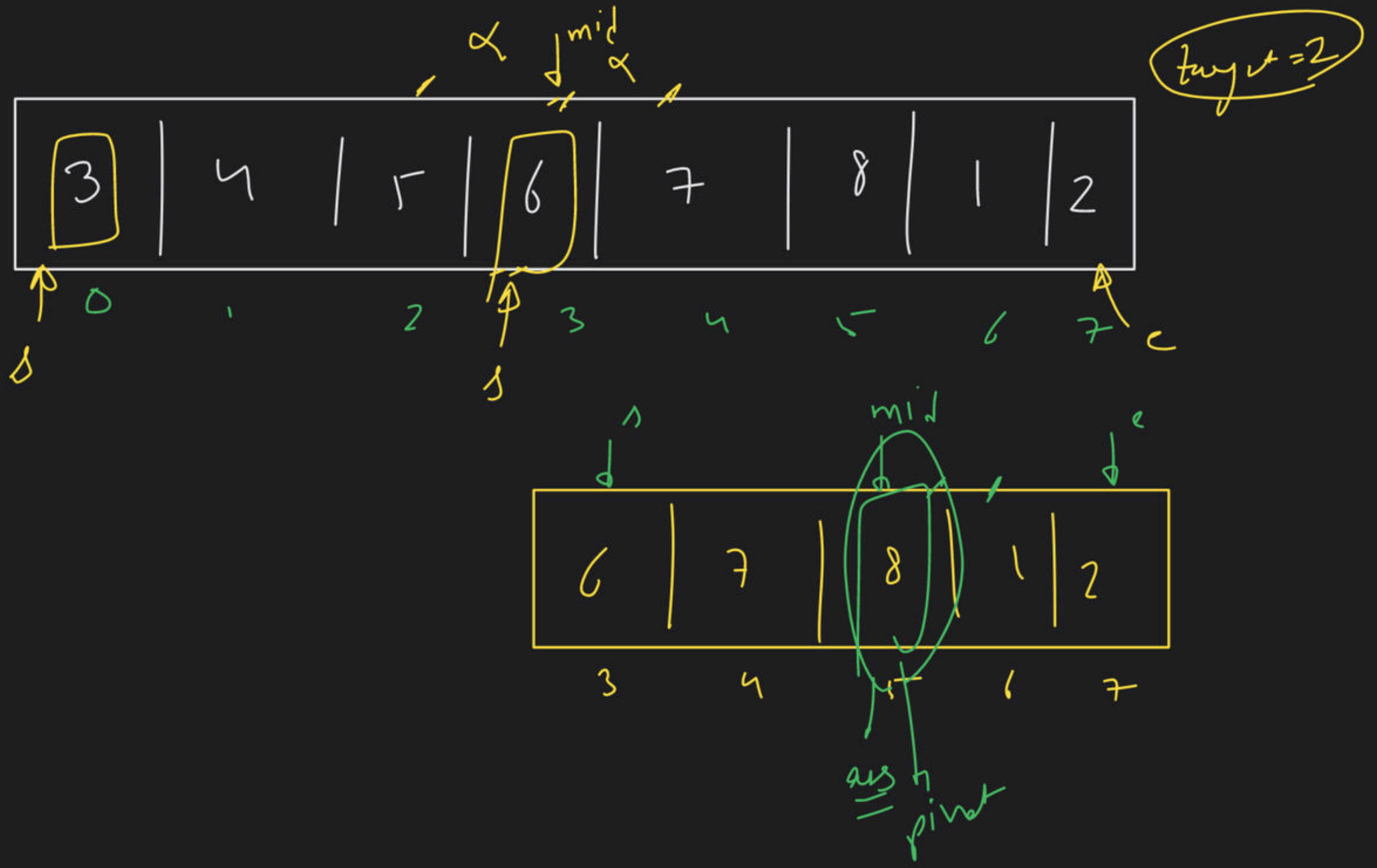

0 | 1 | 2 | 3 | 4 | 5 | 6
6 | 0 | 1 | 2 | 3 | 4 | 5 |
5 | 6 | 0 | 1 | 2 | 3 | 4
4 | 5 | 6 | 0 | 1 | 2 | 3

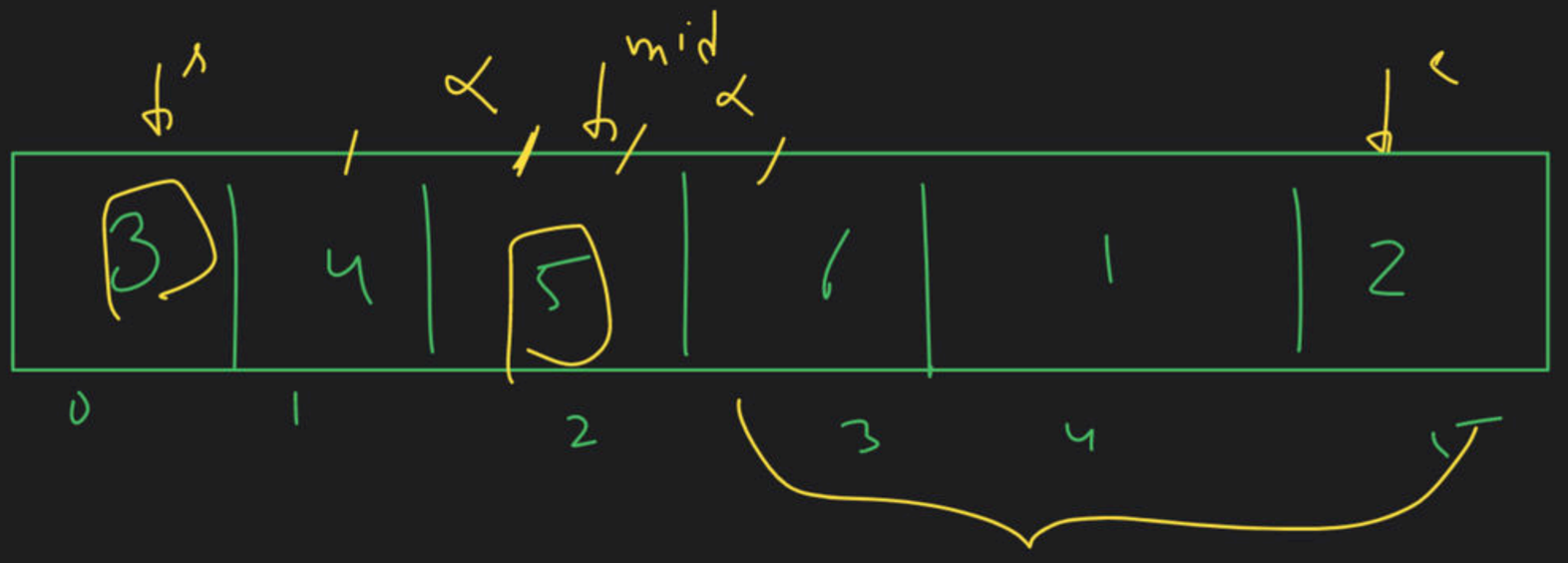

Sorted
pivot
Sorted
target = 2
4
5
6
0
1
2
3
0
1
2
3
4
5
6
Pivot
sorted → 0 → pivot
sorted 2 → pivot + 1 → n-1
6
5
4
3
2
1
0


pivot
pivot+1
n-1
4
5
6
7
0
1
2
3
arr
0 → n-1
0
1
2
3
4
5
6
7
array A ←
0 → pivot
B.S
array B ←
(pivot+1) → (n-1)
target = 2
B.S
Solved
4 2 2
0 3
2

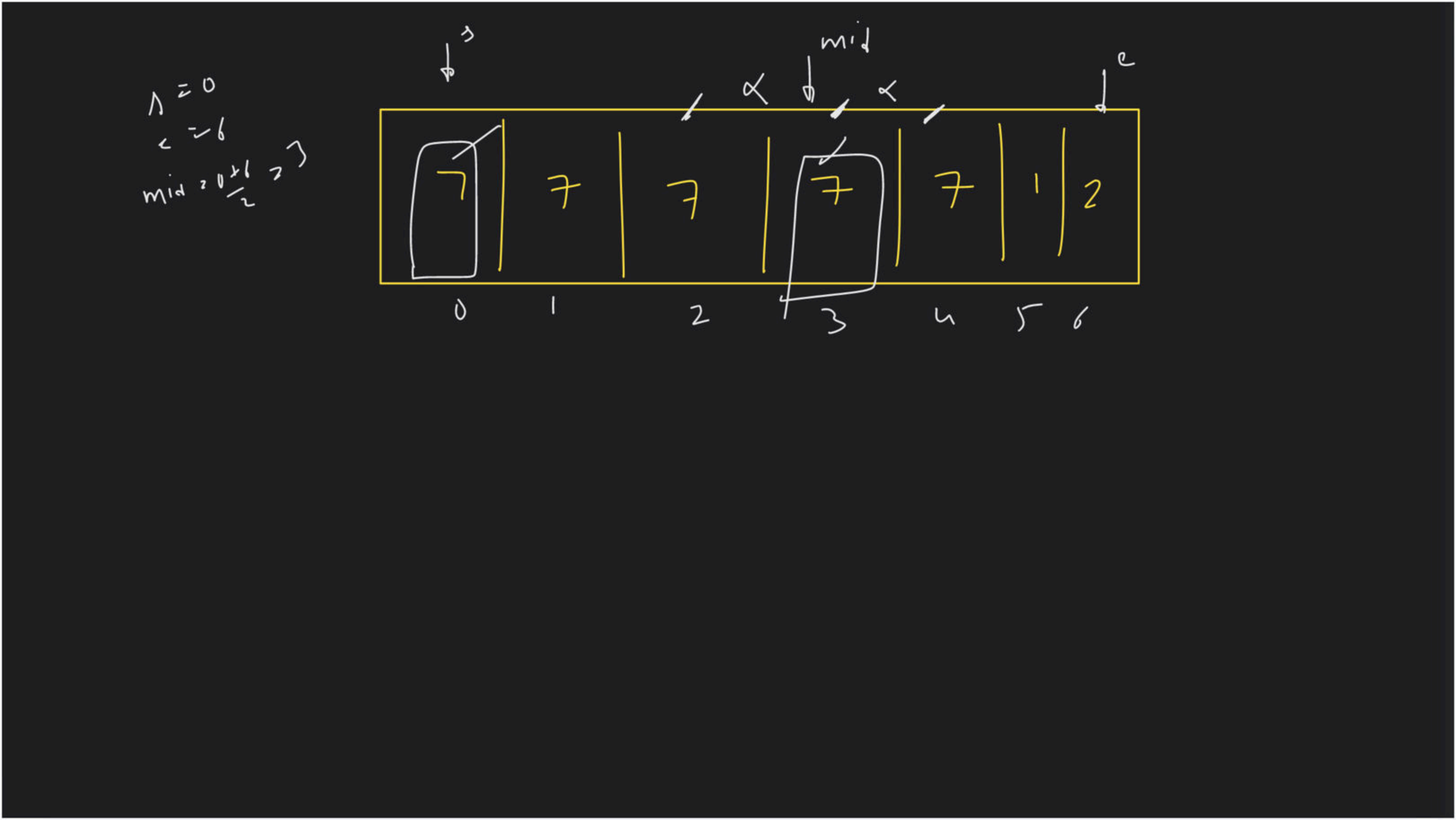

mid
1
s
c

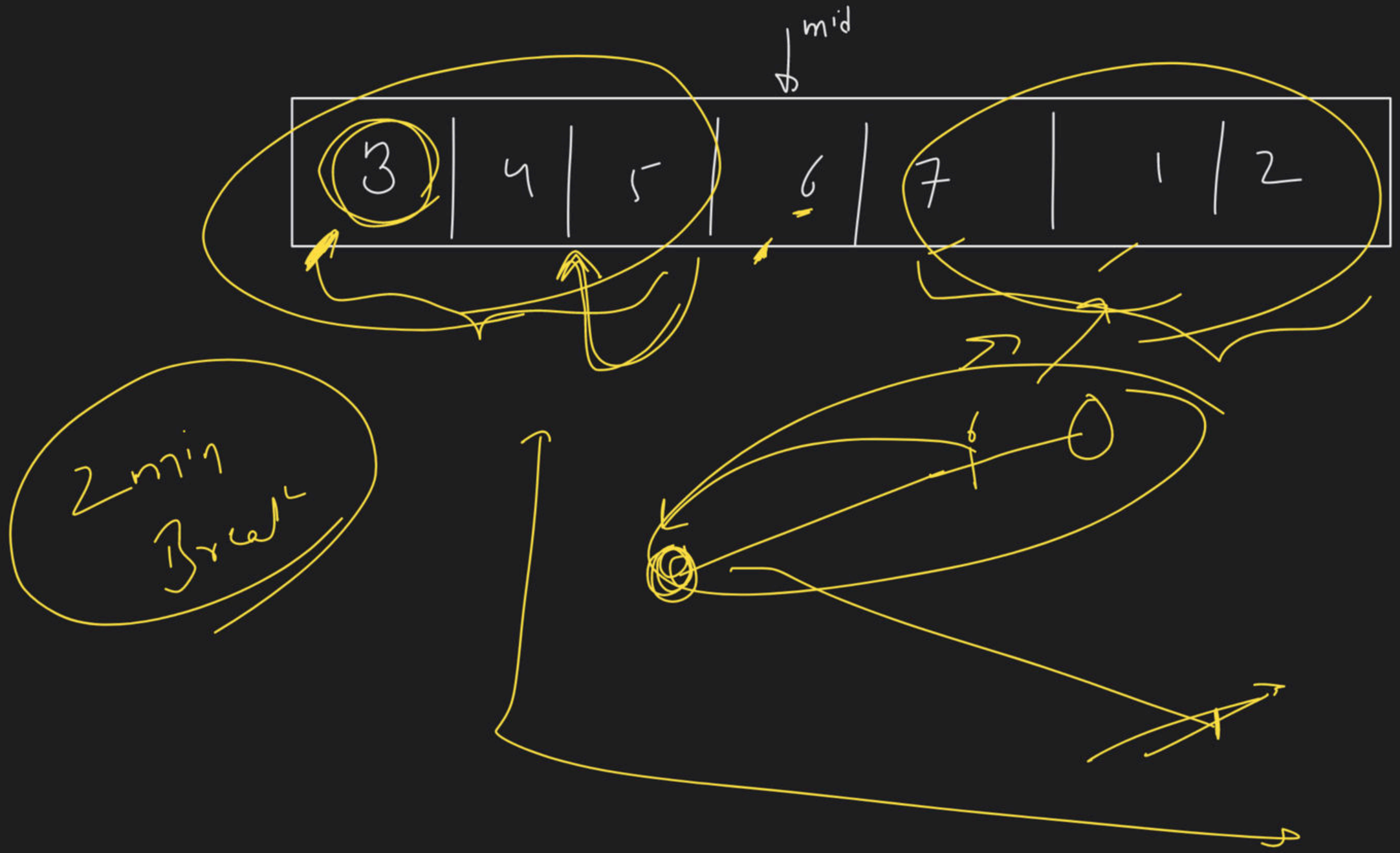

s = 0
e = 0
mid = (s+e)/2
mid-1
mid
mid+1
5
s
e
s = e
return s;
return -1;
if (arr[i] >= arr(mid)
// left search
e = mid-1
else
s = mid
mid+1
while(s<e)
return s;
5
s
e

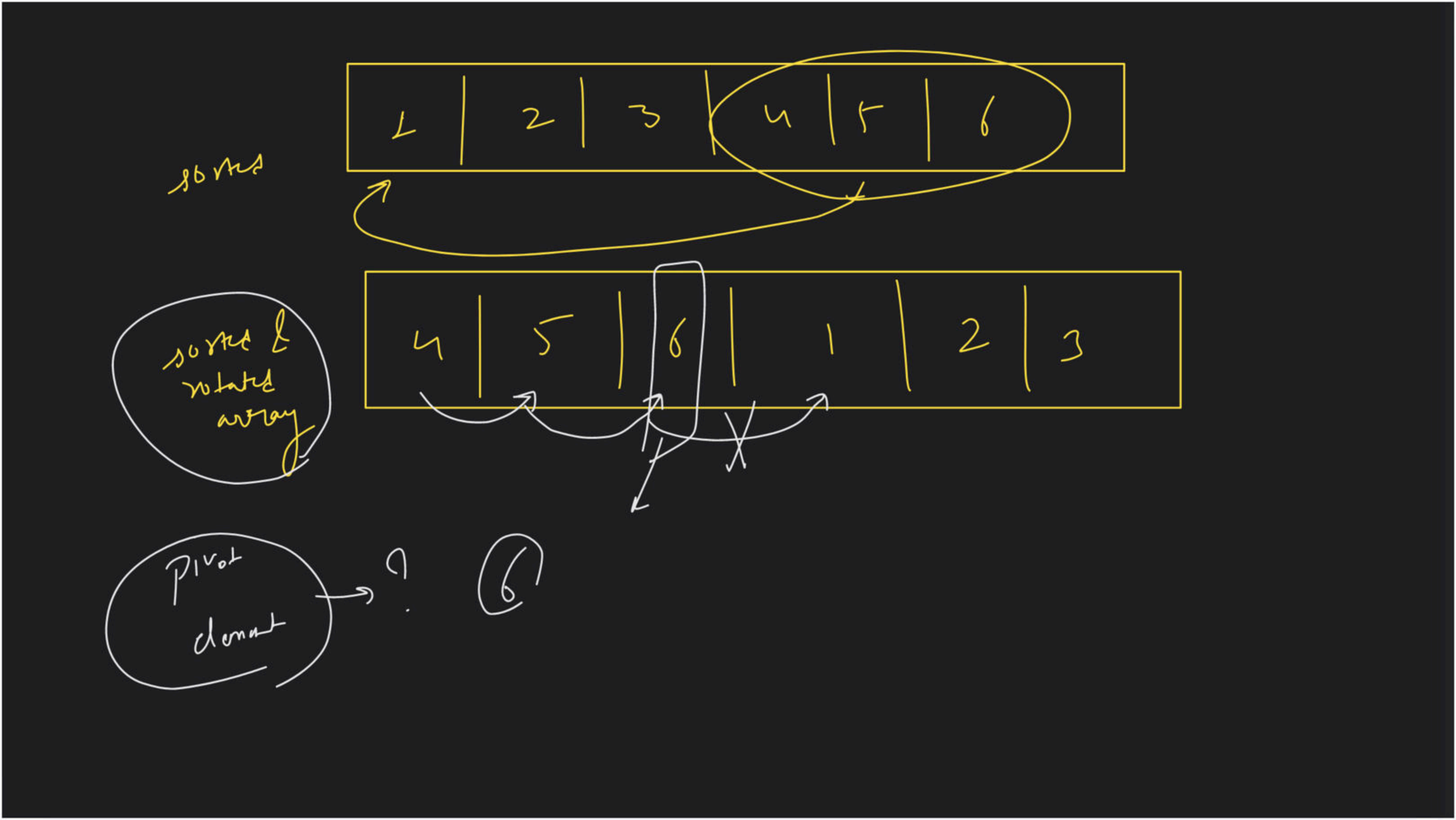

mid
3
4
5
6
7
8
1
2
0
1
2
3
4
5
6
7
mid
6
7
8
1
2
3
4
5
6
7
aus
pivot

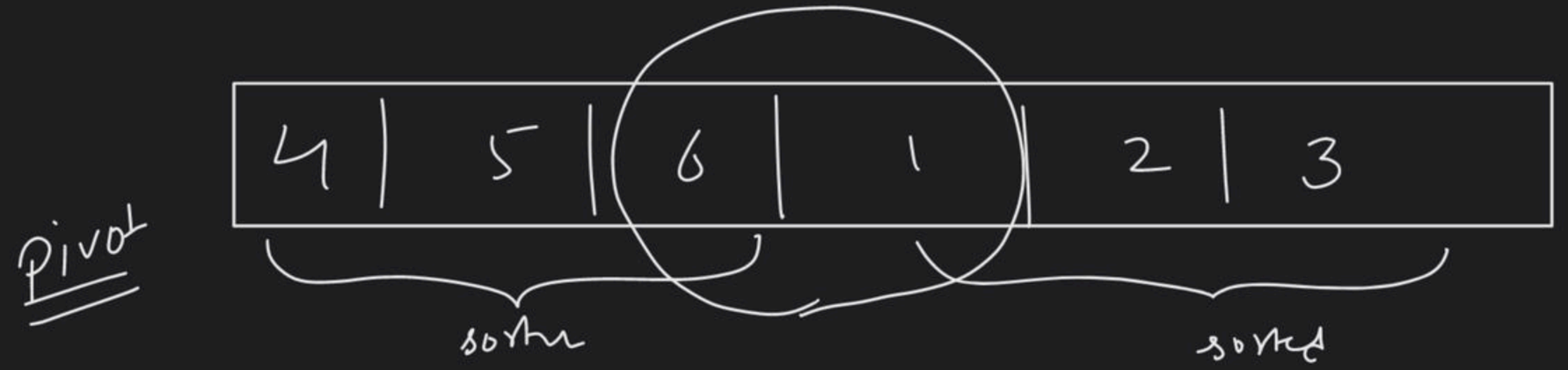

l
mid
r
3
4
5
6
1
2
0
1
2
3
4
5

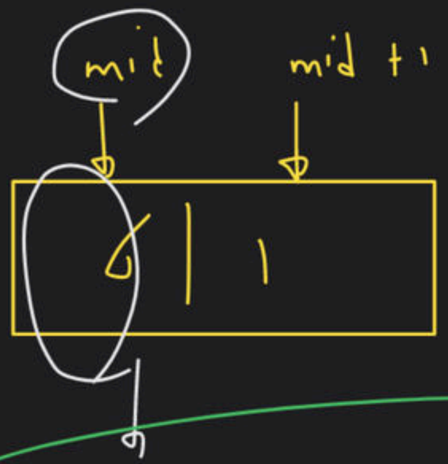

5
6
1


s
mid
e
7
7
7
7
7
1
2
0
1
2
3
4
5
6

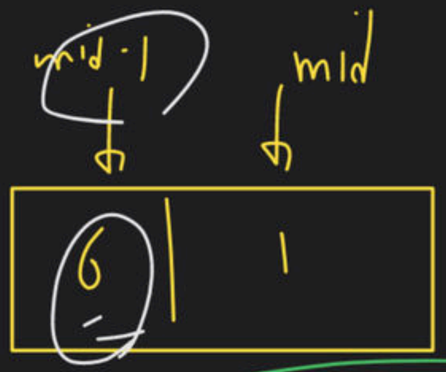

mid
3
4
5
6
7
1
2
2 min
Break
0


sorted
1 | 2 | 3 | 4 | 5 | 6
sorted & rotated array
4 | 5 | 6 | 1 | 2 | 3
pivot element
?
6

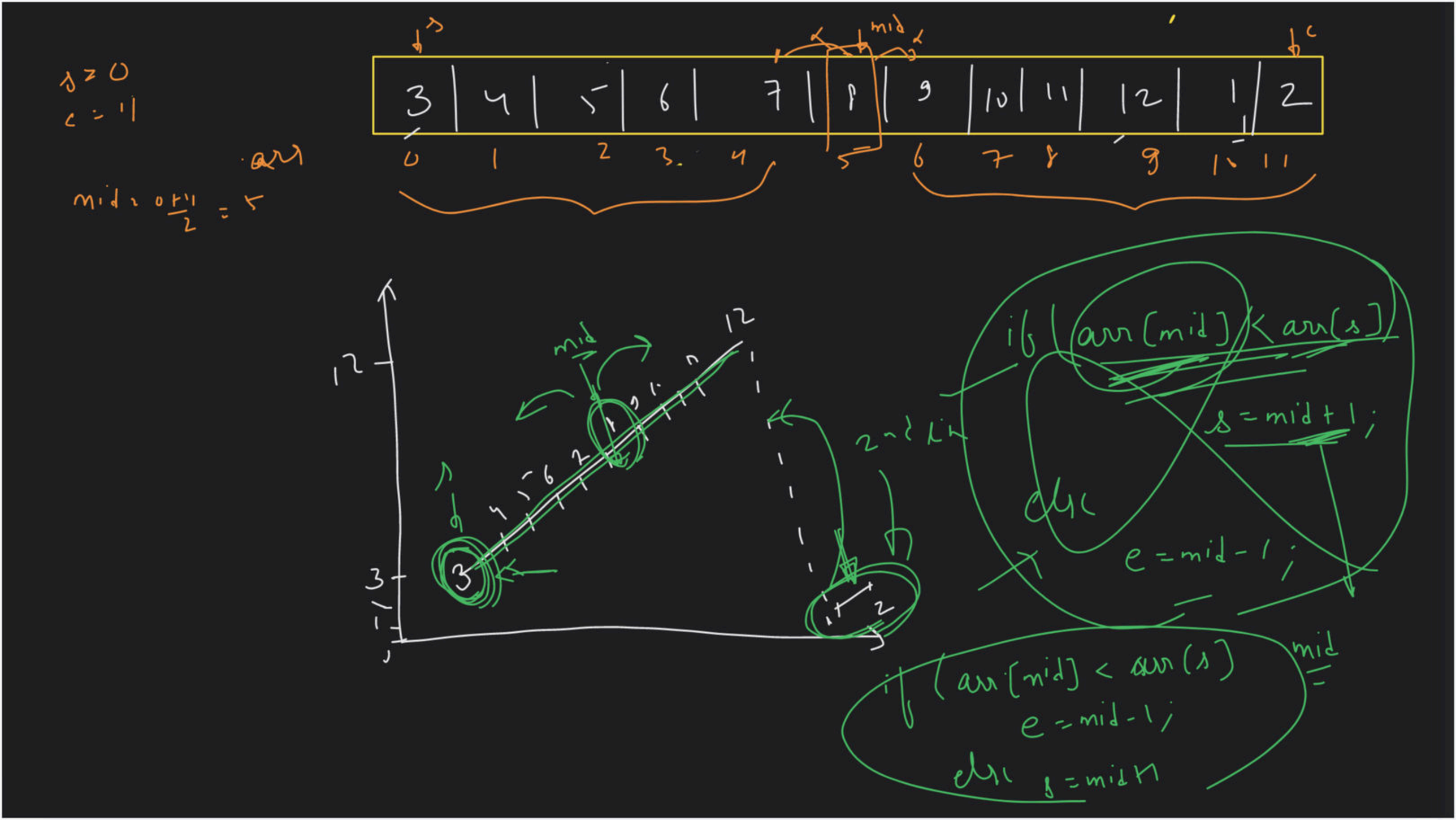


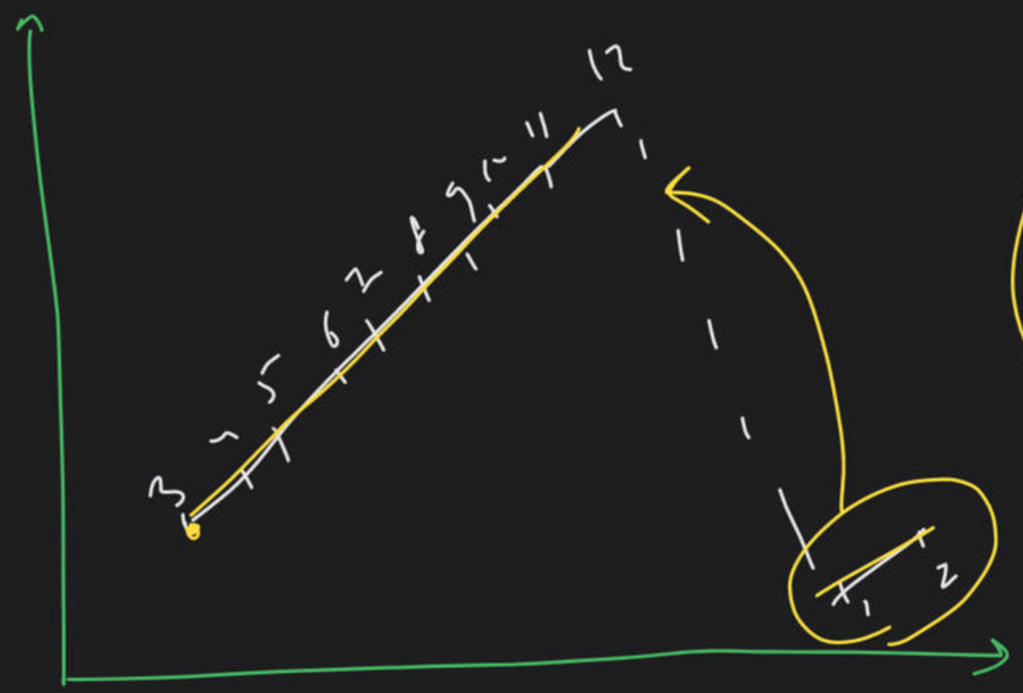


if (arr[mid] > arr[mid+1])
return mid

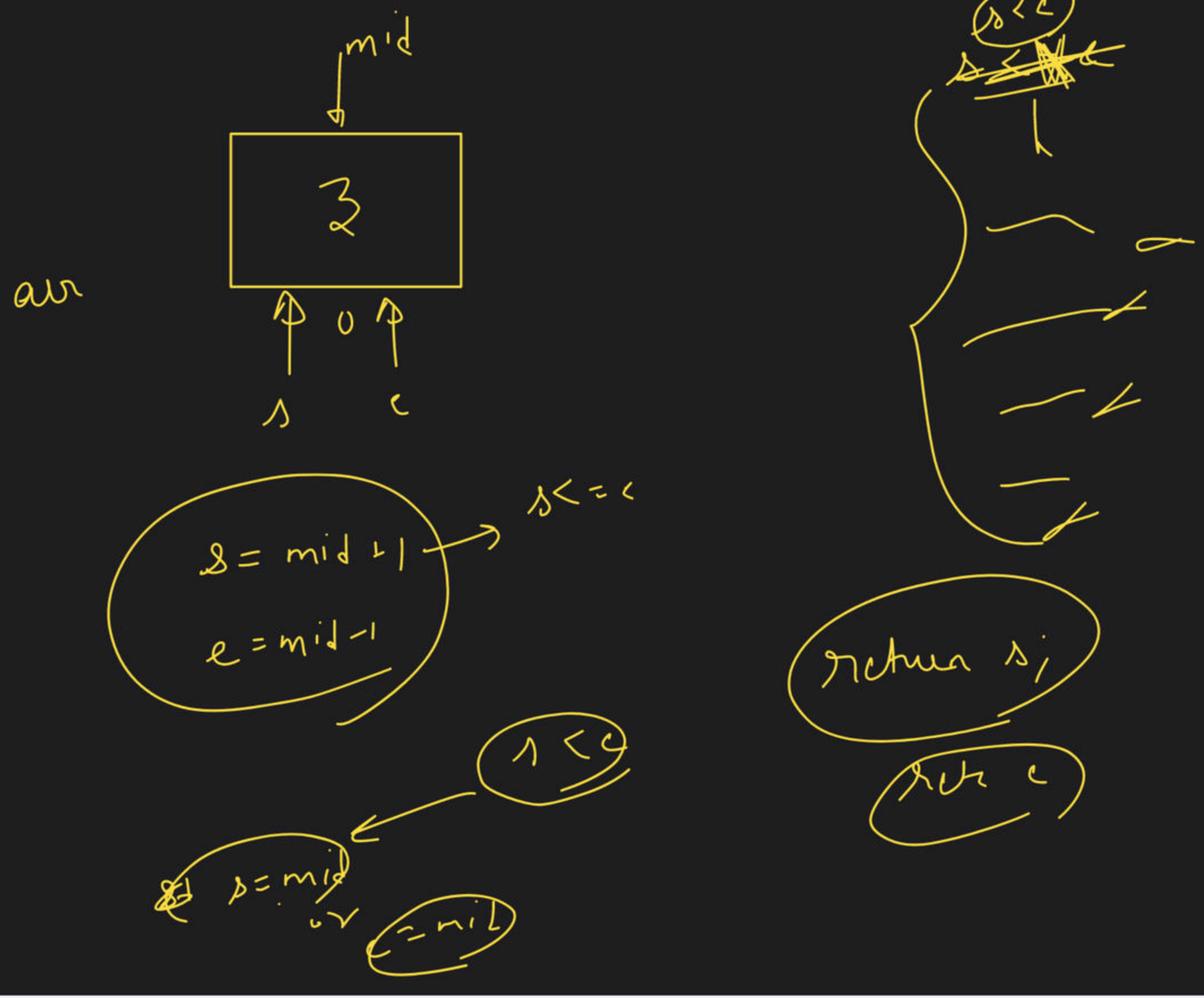


if (arr[mid-1] > arr[mid])
return(mid-1);

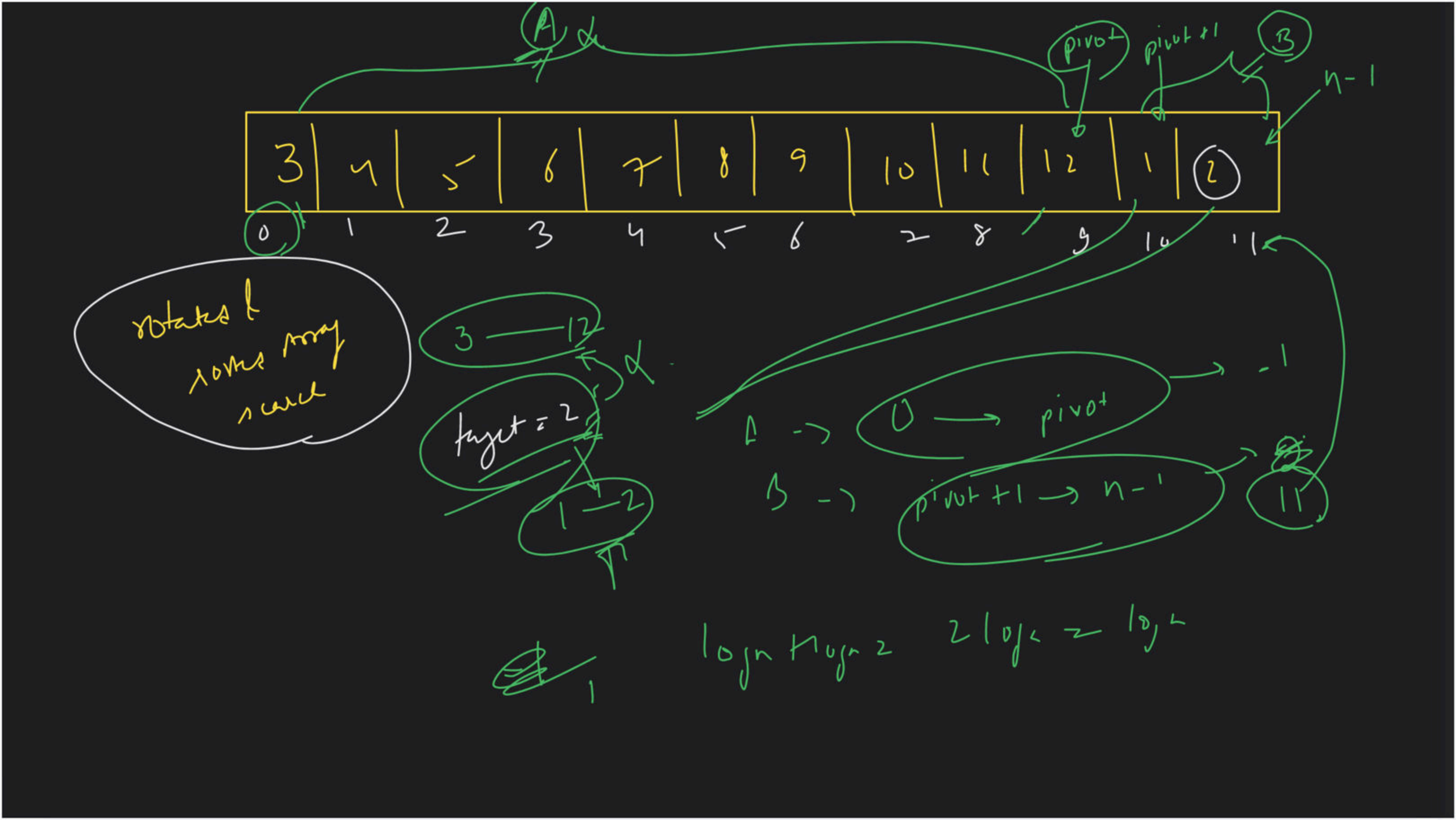

s=0
e=11
arr
mid = (s+e)/2 = 5
s
mid
e
3 4 5 6 7 8 9 10 11 12 1 2
0 1 2 3 4 5 6 7 8 9 10 11
12
3
mid
12
3
1 2
if (arr[mid] < arr[s])
s = mid + 1;
else
e = mid - 1;
if (arr[mid] < arr(s)
e = mid - 1;
else s = mid+1
mid

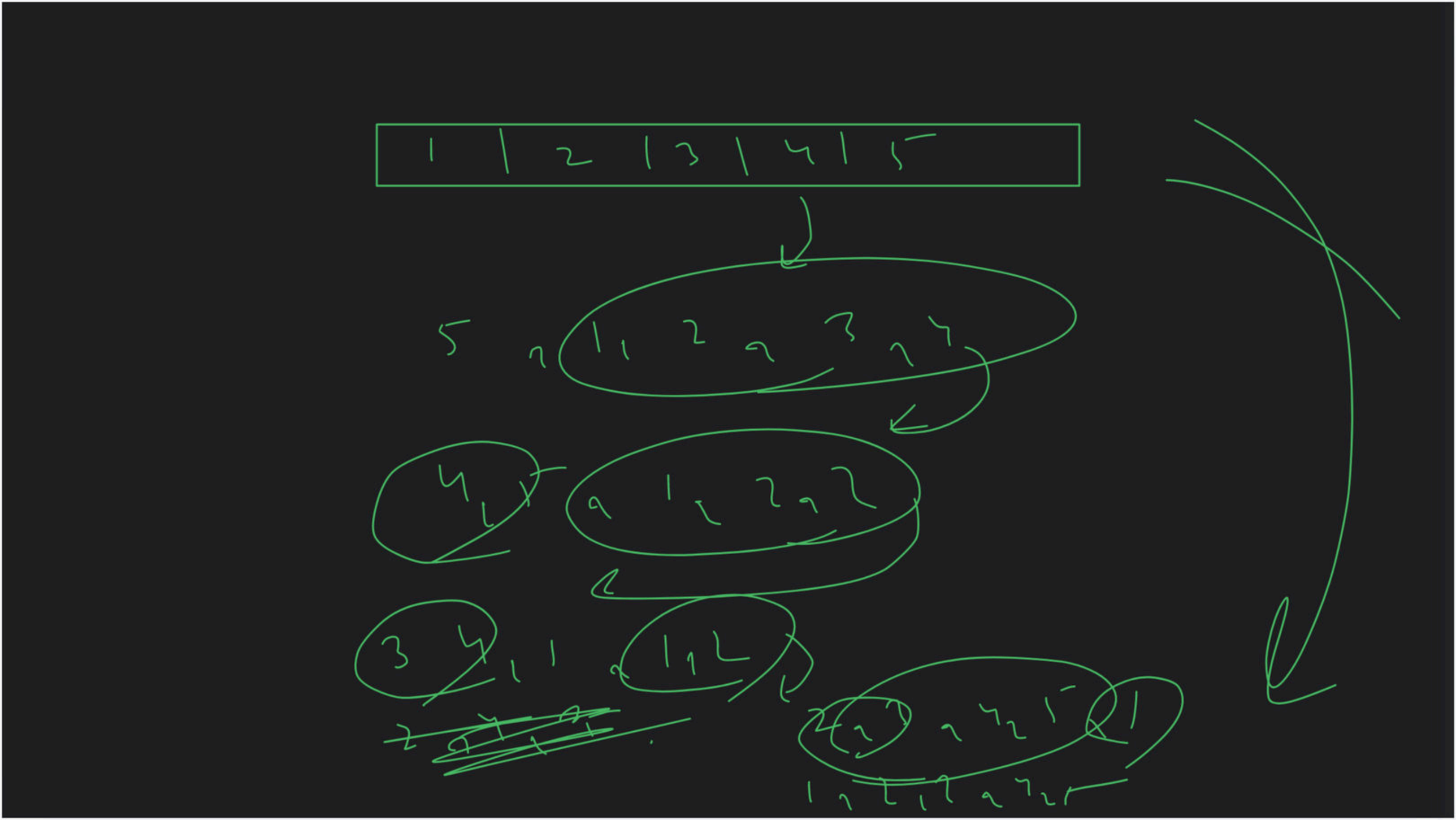



```
if (arr[mid] < arr[s])
    e = mid - 1;
else
    s = mid + 1;
```


mid
3
aur
s
e
s<=e
s = mid + 1
e = mid - 1
s<e
s = mid
or
= mid
return s;
return e


7
6
5
4
3
2
1


A
pivot
pivot+1
B
n-1
3 4 5 6 7 8 9 10 11 12 1 2
0 1 2 3 4 5 6 2 8 9 10 11
rotated k
times array
search
3 — 12
target = 2
1 — 2
A -> 0 -> pivot -> -1
B -> pivot+1 -> n-1
11
1
log n + log n/2
2 log n = log n



→ Square Root of a number using Binary Search

Pivot & Search in Rotated Sorted Array

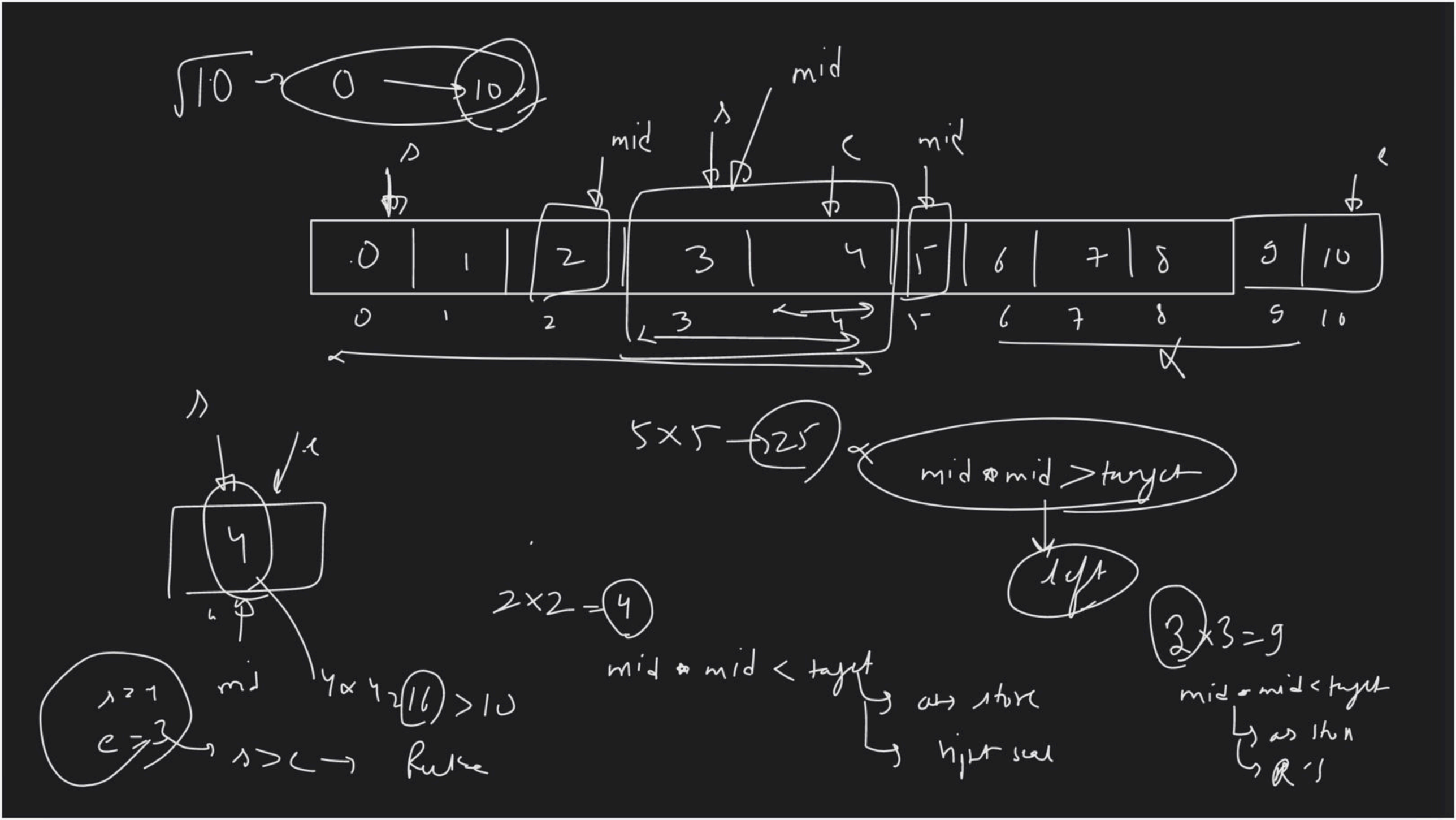
$\sqrt{12}$ → 0 ——→ 12

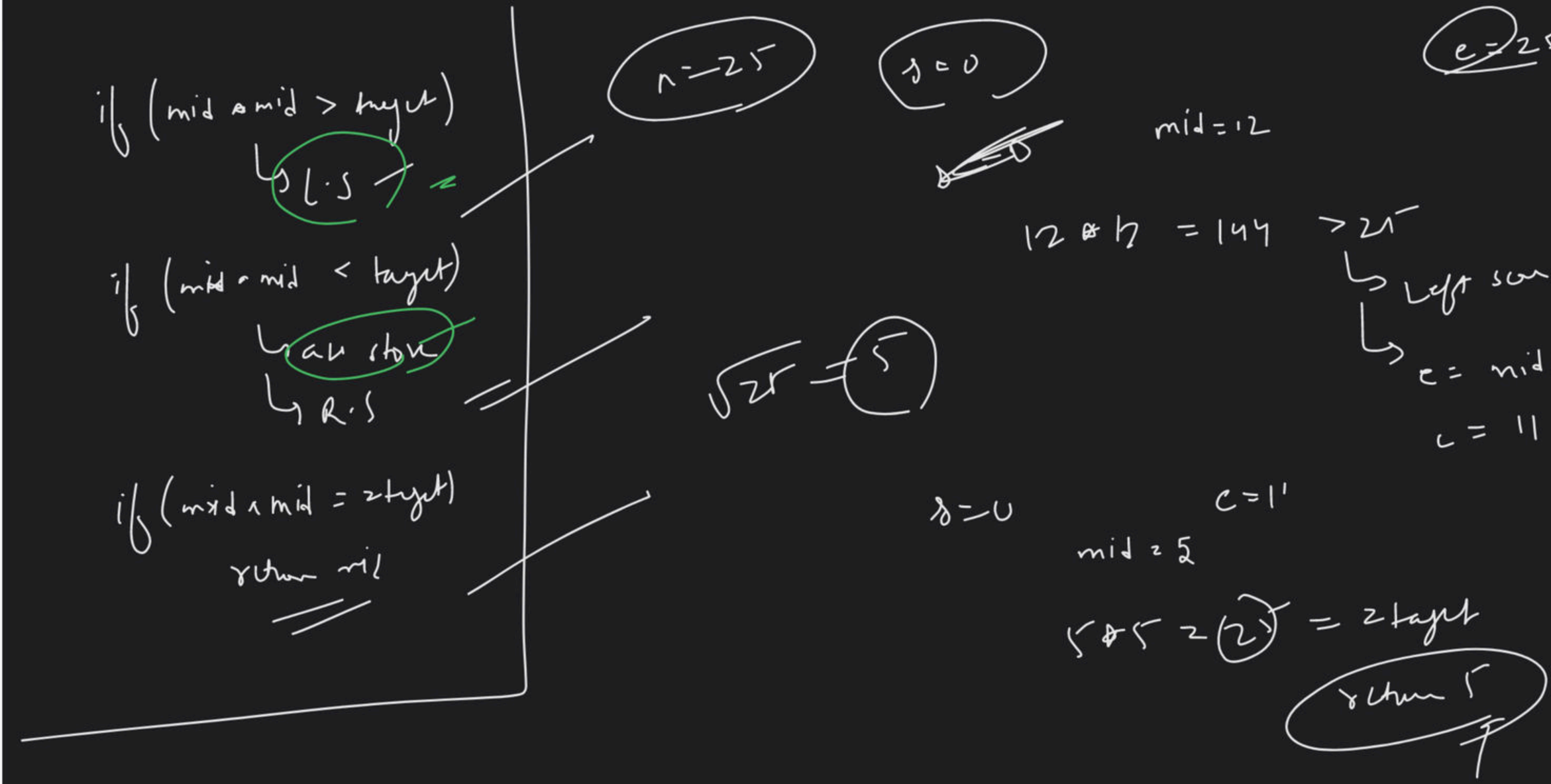
$\sqrt{8}$ → 0 ——→ 8

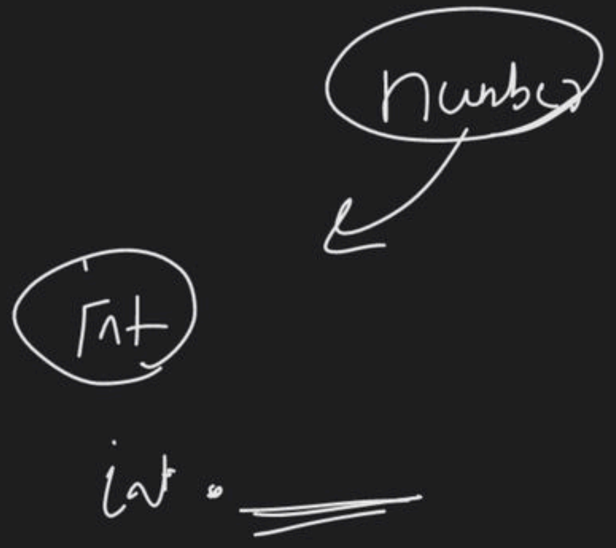
$\sqrt{50}$ → 0 ——→ 50

$\sqrt{100}$ → 0 ——→ 100

search space


√10
0
10
mid
mid
mid
0 1 2 3 4 5 6 7 8 9 10
0 1 2 3 4 5 6 7 8 9 10
5×5 = 25
mid * mid > target
left
2×2 = 4
mid * mid < target
go right
right side
3×3 = 9
mid * mid < target
go right
R-1
4
mid
4×4 = 16 > 10
Rules


if (mid * mid > target)
L.S = mid - 1
if (mid * mid < target)
ans store
L.S = mid + 1
if (mid * mid = = target)
return mid
l = 0
r = 25
mid = 12
12 * 12 = 144 > 25
Left side
r = mid - 1
r = 11
√25 = 5
l = 0
r = 11
mid = 5
5 * 5 = 25 = target
return 5



25 → 5

26 → 5.

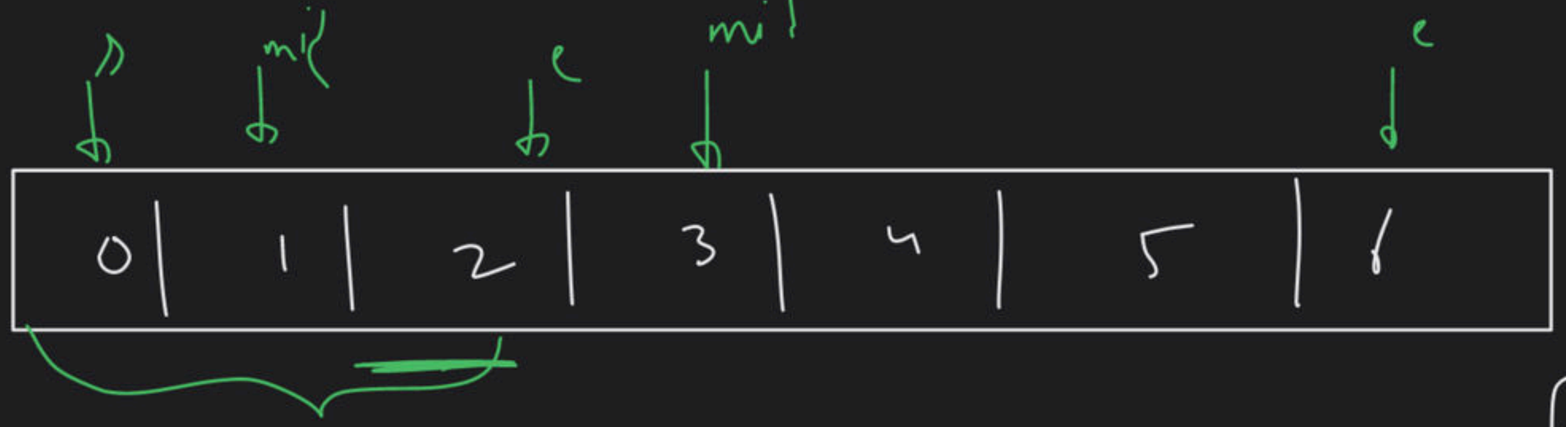
27 → 5.

28 → 5

not Audible

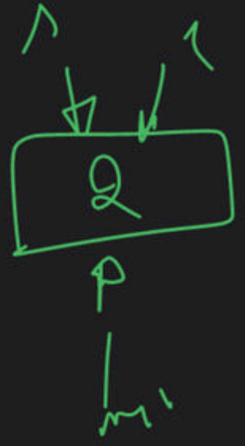
Reload

target = 6



$3 * 3 = 9 > 6$

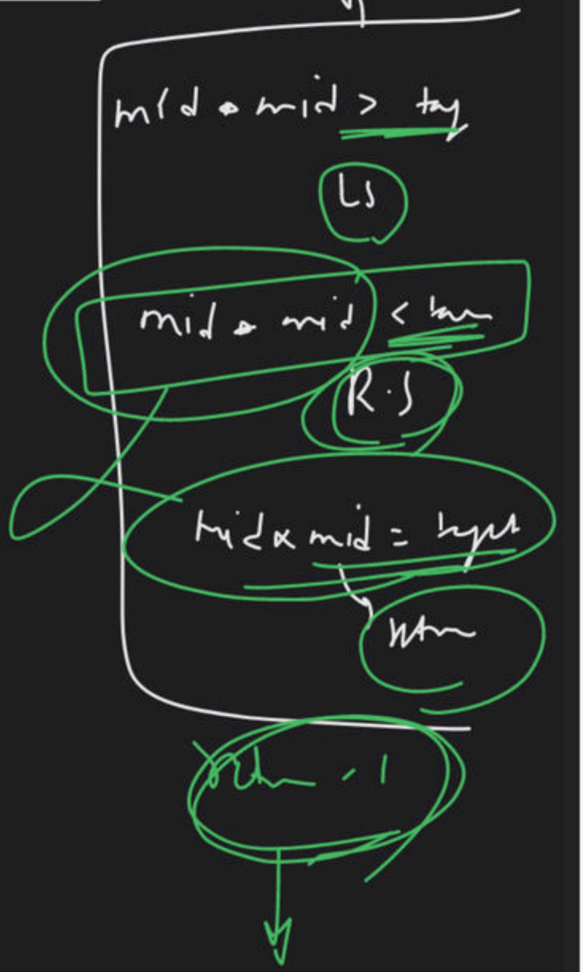
$|\phi| = 1 < 6$

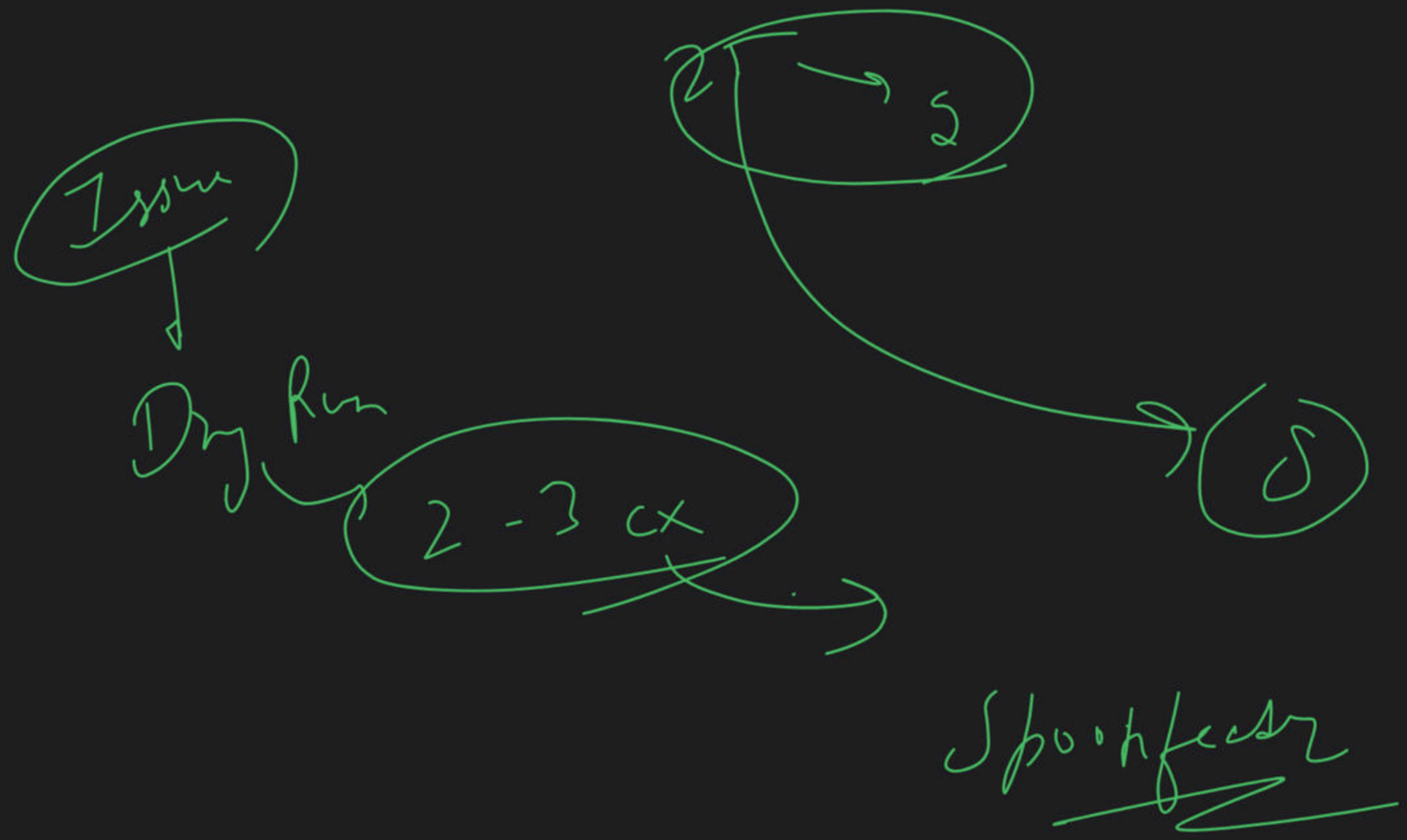


$2 * 2 > 4 < 6$

$s = 3$
$e = 2$
→ Rokdia

By a slide




Issue
Dry Run
2 - 3 cx
2 → 5
5
Smoothness

find floating

$\sqrt{10}$ = 3.16

Hour

ans = 3

0.1

3.1 × 3.1 <= 10 → true

3.2 → 3.2 × 3.1 > 10 → false

3.3

3.4

3.5

3.6

3.7

3.8

3.9

3 / 0.1 / 3.1 / 0.1 / 3.2

3.1 / 0.01 / 3.11 / 0.01 / 3.12

0.01

ans = 3.1

3.11 → 3.11 × 3.11 <= 10 → true

3.12 → 3.11 × 3.12 <= 10

3.13 → 3.13 × 3.13 <= 10

3.14 →

3.15

3.16

3.17 → false

3.18

3.19

3.161

3.162

0.001

Binary search

3.16

0.001

3.161

3.161

3.162

3.163

3.164

Break for

0.0001

0.00001

0.000001

finalAns = 3;
step = 0.1

```
for (int i = 0; i < precision; i++)
{
    for (double j = finalAns; j*j <= n; j = j + step)
    {
        finalAns = j; // ans store
    }
    step = step / 10;
}
```

finalAns: ~~3.1~~ ~~3.11~~ ~~3.12~~ ~~3.13~~ 3.11

step: ~~0.1~~ 0.01

n = 10

j = 3
3 * 3 = 9 <= 10 → true

j = 3 + 0.1 = 3.1
3.1 x 3.1 = 9.61 <= 10 → true

j : 3.1 + 0.1 = 3.2
3.2 x 3.2 = 10.24 <= 10 → false

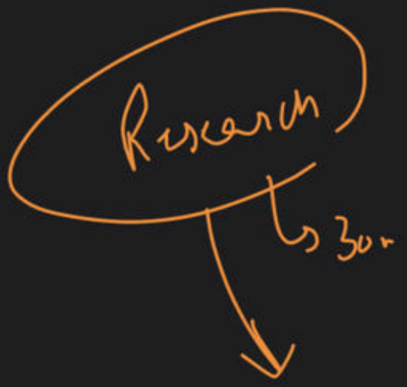
step = 0.1 / 10 = 0.01

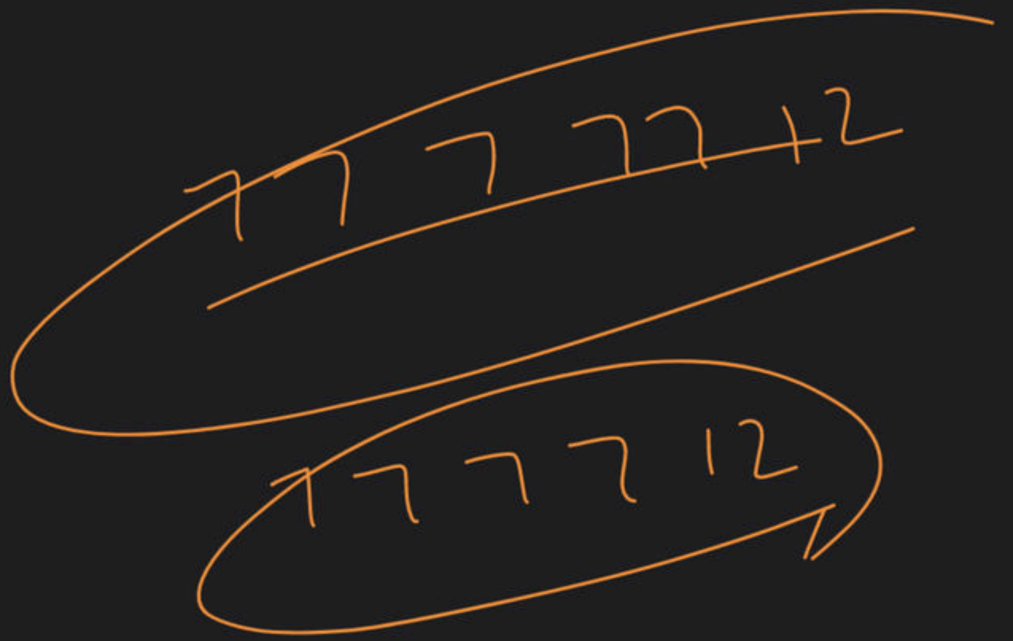
j = 3.1
3.1 x 3.1 = 9.61 <= 10 → true
j = 3.1 + 0.01 → 3.11
3.11 x 3.11 = 9.6721 <= 10 → true

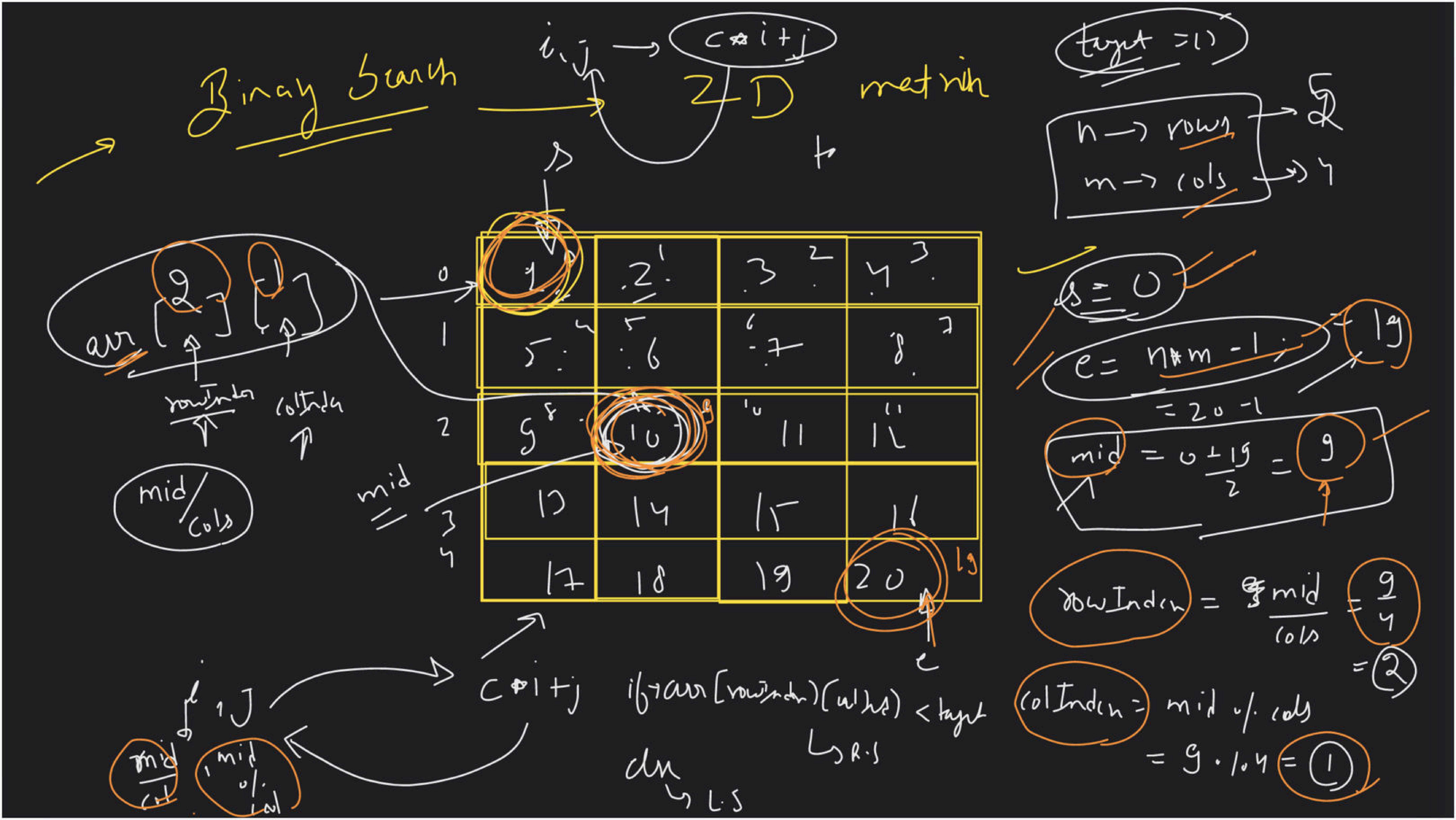
j = 3.11 + 0.01 = 3.12
3.12 x 3.12 → 9.7344 <= 10 yes

j = 3.12 + 0.01 = 3.13
3.13 x 3.13 → 9.7969 <= 10

```
for (int i=0 ; i<precision ; i++)
{
    for (double j=finalAns ; j*j<=n ; j+=step)
    {
        finalAns=j;
    }
    step = step/10;
}
```

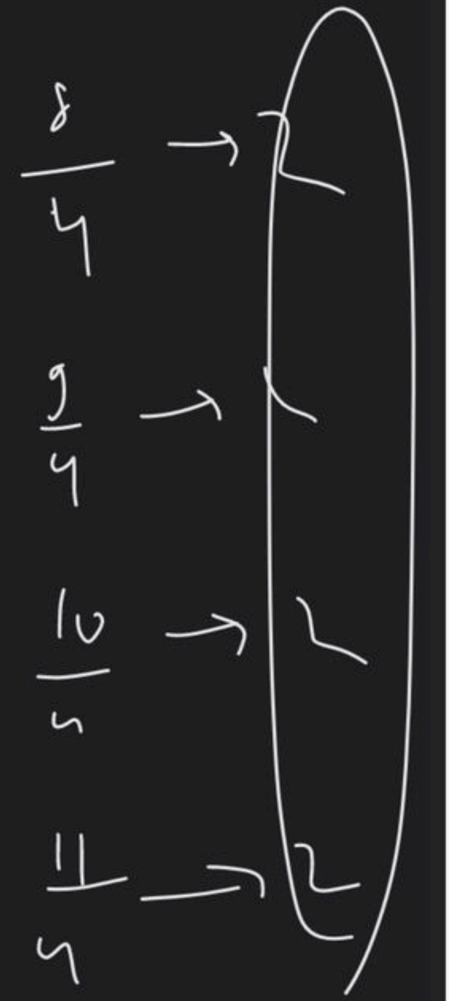


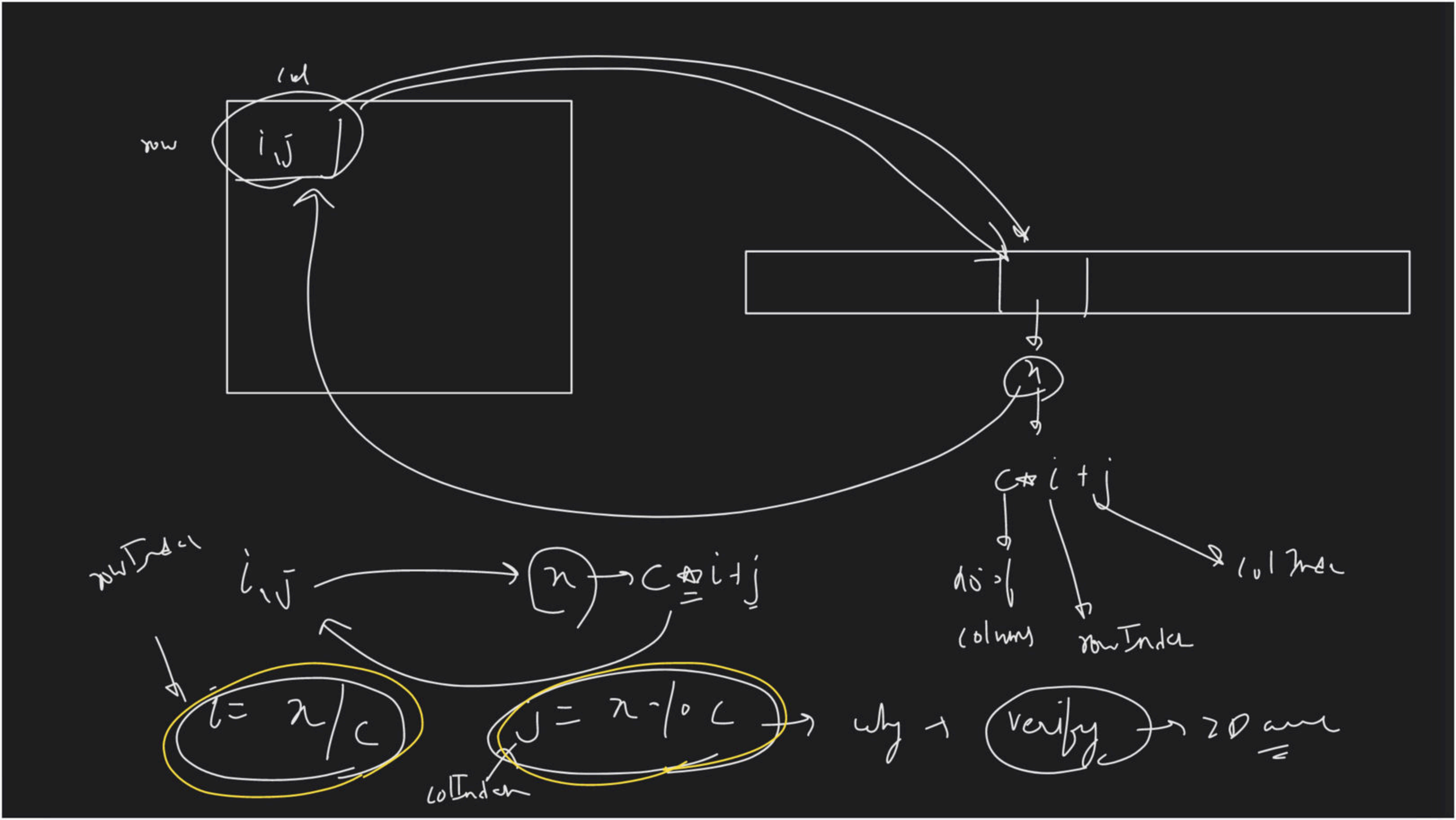


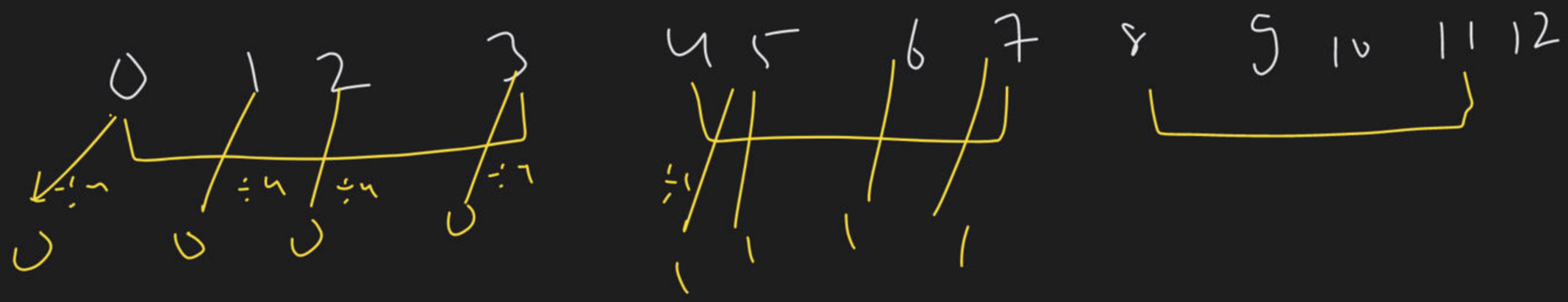


Binary Search
2D matrix
i, j → c*i+j
target = 11
n → rows → 5
m → cols → 4
s = 0
e = n*m - 1
= 20 - 1
19
mid = 0+19 / 2 = 9
rowIndex = mid / cols = 9 / 4 = 2
colIndex = mid % cols
= 9 % 4 = 1
arr [2] [1]
rowIndex
colIndex
mid / cols
mid
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
c*i+j
if arr[rowIndex][colIndex] < target
R.S
else
L.S
mid / col
mid % col

```
int s = 0;
int totalsize = rows * cols;
int e = totalsize - 1;
int mid = s + e / 2;

while( s <= e)
{
    rowIndex = mid / cols = mid / m;
    colIndex = mid % m;
    int element = arr[rowIndex][colIndex]

    if (element == target)
        return true;

    if (element > target)
        ↳ left → e = mid - 1

    else
        ↳ R.S → s = mid + 1
```




col
row
i, j
n
C* i + j
rowIndex
i, j
n
C* i + j
0 -> columns
rowIndex
colIndex
i = n / c
j = n - 0 c
colIndex
why
verify
2D array


0 1 2 3 4 5 6 7 8 9 10 11 12

① Search in a nearly sorted array in log time

② Division of 2 no's using Binary Search

③ find odd occuring element using Binary Search