- Finding mid
- Q1. Search in Rotated Sorted Array
- Q2. Finding square root of N
- Q3. Median of 2 sorted arrays

9 Dec → 23 Dec

25 Dec → holiday

27 Dec

3 Jan → class

Best practice to compute Mid

Let's assume that we have a datatype → dtype which has a range -100 to 100.

Array of length 100 ⇒ 0 to 99 indices

① dtype $l = 0$, $r = 99$

dtype $mid = \frac{l+r}{2} = \frac{0+99}{2} = 49$

⇒ go right $l = mid + 1$

② $l = 50$ $r = 99$

$mid = \frac{l+r}{2} = \frac{50+99}{2} = \frac{149}{2}$ ← overflow

$Mid = \frac{l+r}{2} =$ **Mid** $l + \frac{(r-l)}{2}$ ✓

↓

$l + \frac{r}{2} - \frac{l}{2} = \frac{l}{2} + \frac{r}{2}$

$= \frac{(l+r)}{2}$

$l = 50$ $r = 99$

$mid = l + \frac{(r-l)}{2} = 50 + \frac{(99-50)}{2} = 50 + 24$

$= 74$

1. Given an array of unique elements which was initially sorted, but someone rotated it at an unknown index, so it is a rotated sorted array.

Given k, check if k is present in array or not.

4 5 6 7 8 9 1 2 3

K = 8 true

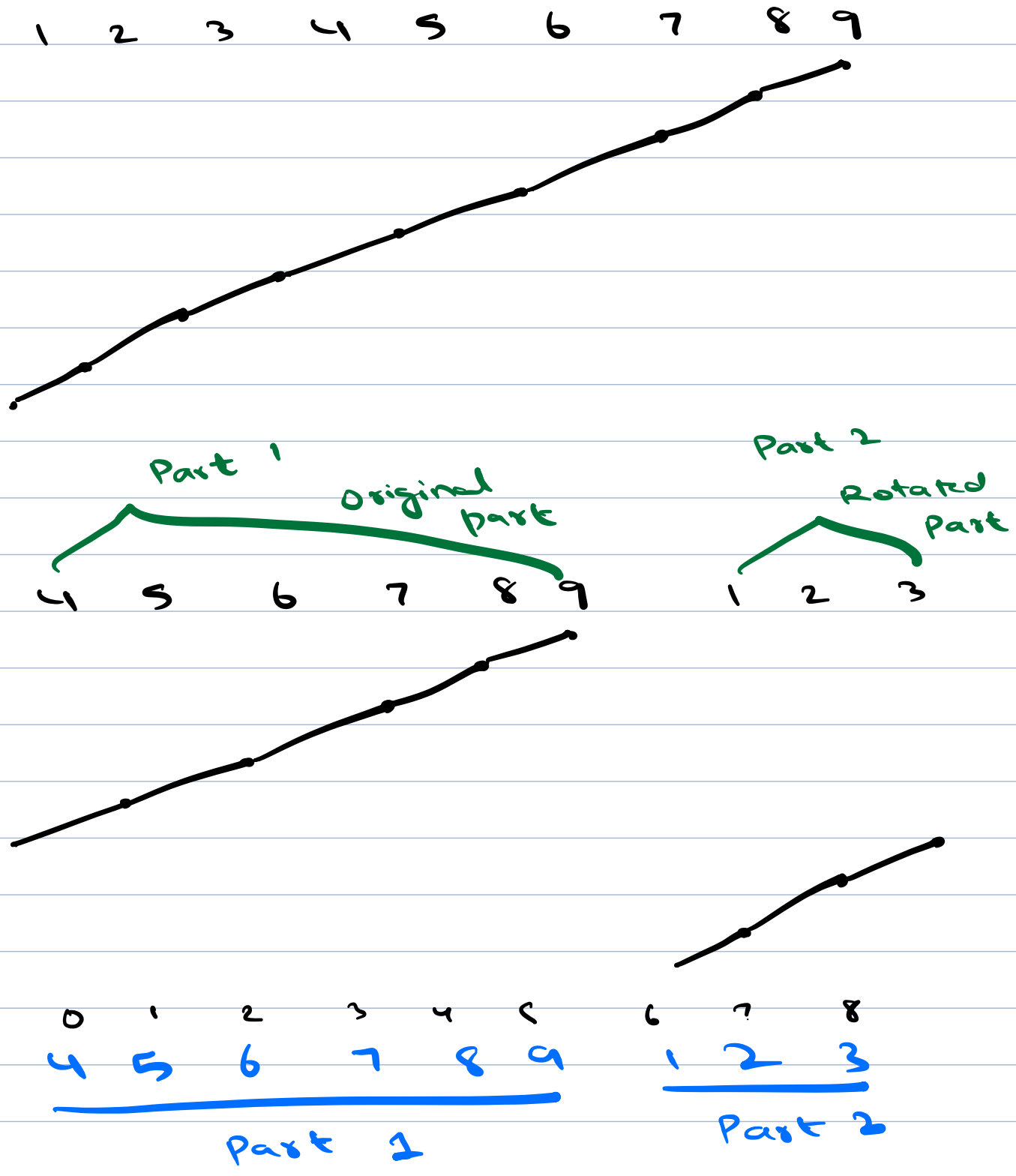
K = 11 false

Brute Force: Do a linear search

TC: O(N) SC: O(1)

K=2

0	1	2	3	4	5	6	7	8
4	5	6	7	8	9	1	2	3

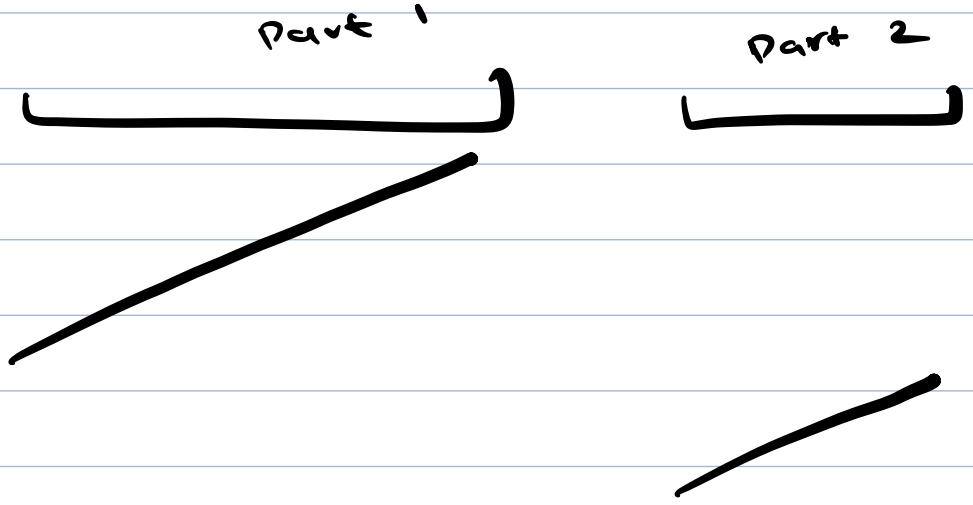


If $a_c < 4$ → Part 2

else Part 1

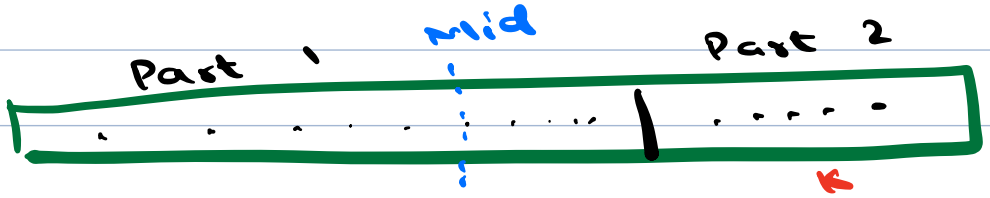
If $k < A[0]$ → Part 2

else Part 1



① Mid is in first part

② K is in second part



Go right, $l = mid + 1$

③ K is in first part

compare to normal side of BS

② mid is in second part

② K is in first part

Go left r = mid - 1

② K is in second part

compare to normal rule of BS

A = 4, 5, 6, 7, 0, 1, 2 (0 1 2 3 4 5 6) K = 0

Part 1 Part 2

l	r	mid	mid is in part?	K is in part?	move?
0	6	3	Part 1	Part 2	Go right l = mid+1
4	6	5	Part 2	Part 2	K<a[mid] 0<1 Left r = mid-1
4	4	4	a[mid] = target		

Stop

```
bool isKPresent(int a[], int N, int k) {

    int l = 0, r = N-1
    while (l <= r) {

        int mid = l + (r-l)/2

        if (A[mid] == k) return true

        if (A[mid] < A[0]) {
                                // Mid is in part 2

            if (k < A[0]) { // k is in part 2
                if (k < A[mid])          // Normal BS
                    r = mid-1

                else
                    l = mid+1
            }
            else {    // k is in part 1
                // left    r = mid-1
            }
        }
        else { // Mid is in part 1

            if (k < A[0]) { // k is in part 2
                // right    l = mid+1
            }
```

else { // k is in part 1

if (k < A[mid]) // Normal BS

r = mid - 1

else

l = mid + 1

}

}

}

return false

TC : O(log₂N)

2. Given N, find integer part of $\sqrt{N}$.

N = 36 ans = 6 $\sqrt{36}$

N = 50 ans = 7 $\sqrt{50}$

N = 65 ans = 8

$7 \times 7 = 49$

$7. \times 7. = 50$

$8 \times 8 = 64$

$8. \times 8. = 65$

$9 \times 9 = 81$

N = 36

i	i^2
1	→ 1
2	→ 4
3	→ 9
4	→ 16
5	→ 25
6	→ 36

$i^2 = N$ $i^2 < N$

N = 50

i	i^2
1	→ 1
2	→ 4
3	→ 9
4	→ 16
5	→ 25
6	→ 36
7	→ 49
8	→ 64

$i^2 \leq N$

```
int ans
for (i = 1 ; i ≤ N ; i++) {
    if (i*i ≤ N) ans = i
    else break
}
```

return ans TC: $O(\sqrt{N})$

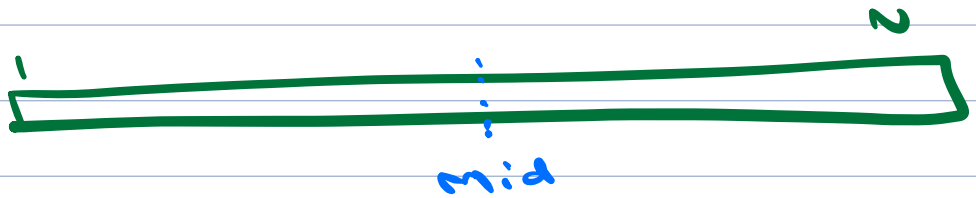
Binary search :

Target : x who is sqrt of N

$\Rightarrow x * x <= N$

Search space : 1 to N

Condition :



① mid * mid = N
return mid

② mid * mid > N
Left x = mid - 1

③ mid * mid < N
ans = mid
Right l = mid + 1

N = 65

l	r	mid	mid^2	
1	65	33	1089 > N	Left
				r = mid - 1

1	32	16	256 > N	Left r = mid-1
1	15	8	64 < N ans = 8	Right l = mid+1
9	15	12	144 > N	Left r = mid-1
9	11	10	100 > N	Left r = mid-1
9	9	9	81 > N	Left r = mid-1
9	8	l > r	stop	

```
int sqrt(int N) {
    l = 1, r = N, ans = 0
    while (l <= r) {
        long int mid = l + (r - l)/2
        if (mid * mid == N)
            return mid
        else if (mid * mid > N)
            r = mid - 1      // left
        else {               // mid * mid < N
            ans = mid
            l = mid + 1      // right
        }
    }
    return ans
}
```

TC : $O(\log_2 N)$

SC : $O(1)$

10:40

N = 65

l r mid mid^2

Median → Middle element in sorted data

N=5 2, 8, 11, 12, 14 Median = 11

N=6 2, 8, 11, 12, 14, 28 Median = $\frac{11+12}{2}$ = 11.5

3. Given 2 sorted arrays A and B, find median of overall merged array.

M = 6
N = 4

A : [1, 3, 4, 7, 10, 12]
B : [2, 3, 6, 15]

1, 2, 3, 3, 4, 6, 7, 10, 12, 15

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

Brute Force : Merge both sorted arrays, and get median

sorted arr[N]

N = 5

① N is odd

Median → arr[N/2]

② N is even

Median → avg of arr[N/2 - 1] and arr[N/2]

TC : O(M + N)

SC : O(M + N)

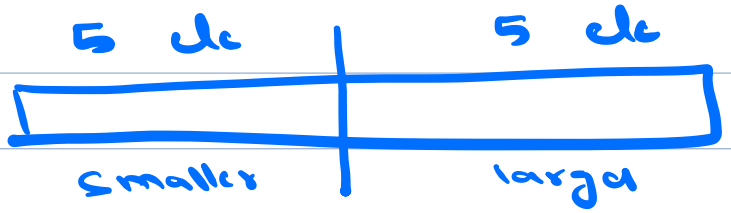
↓

Merge and store in new array

Optimised Approach

A : [1, 3, 4, 7, 10, 12]
B : [2, 3, 6, 15]

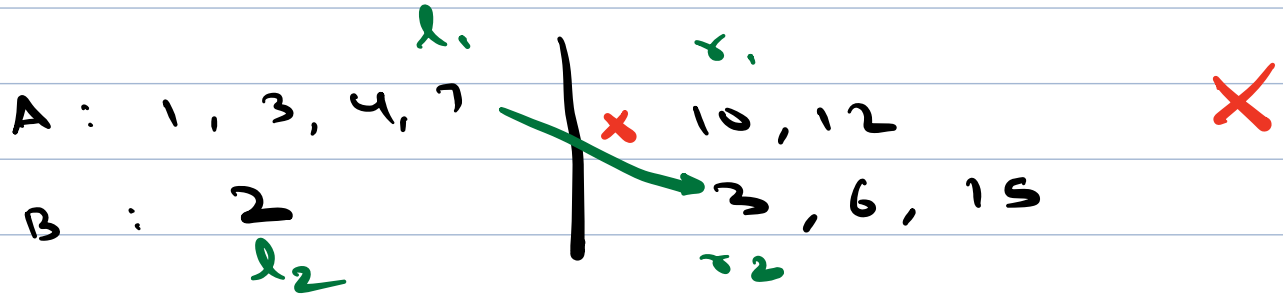
6
4
] → 10 elements

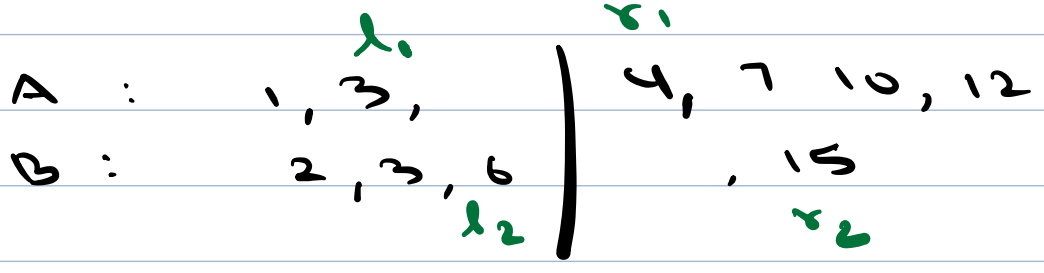


A : [1, 3, 4, 7 10, 12]
B : [2, 3, 6, 15]

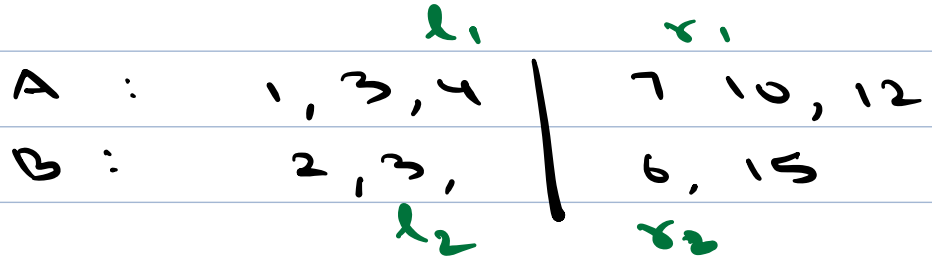
$l_1 < r_2$
and $l_2 < r_1$

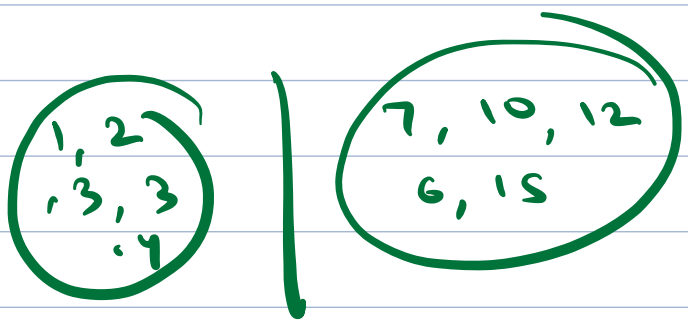
How do I check if partition is valid ?

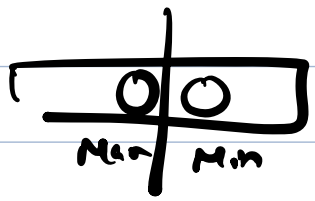




$3 < 15$
$l_1 < r_2$
and
$l_2 < r_1$
$6 \not< 4$





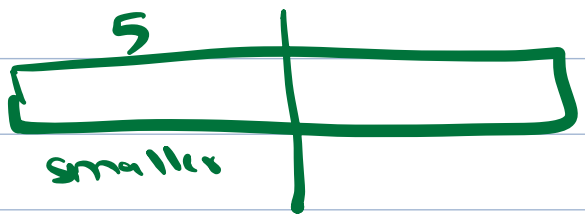


Max
↓
4

Min
↓
6

avg = 5
Median

A : [1, 3, 4, 7 10, 12]
B : [2, 3, 6, 15]



BS : How many elements will I pick from A to put in first half ?

l	r	mid
0	5	2

A : 1, 3, | 4, 7, 10, 12
B : 2, 3, 6 | 15

l_1 r_1 l_2 r_2

$l_1 < r_2$ ✓

and

$l_2 < r_1$

$6 \not< 4$

Invalid partition

$l_2 \not< r_1$

Right $l = mid + 1$

l	r	mid
3	5	4

A : 1, 3, 4, 7 | 10, 12

B : 2, | 3, 6, 15

l_1 r_1 l_2 r_2

$7 \not< 3$

$l_1 < r_2$

and

$l_2 < r_1$

$2 < 10$

$l_1 \not< r_2$

Left $r = mid - 1$

l	r	mid
3	3	3

A : [1, 3, 4, | 7, 10, 12]

B : [2, 3, | 6, 15]

l_1 r_1 l_2 r_2

$l_1 < r_2$ and $l_2 < r_1$

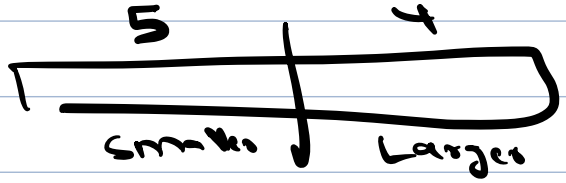
Valid partition

Max $(a[l_1], a[l_2])$	Min $(a[r_1], a[r_2])$
4, 3	min(7, 6)
4	6

Median = avg $= \frac{4+6}{2} = 5$

A + B = 9

Odd → 9



Median = max $(a[l_1], a[l_2])$

TC : O (min (log n, log m))

// assume A is smaller

if (A is larger) swap (A, B)

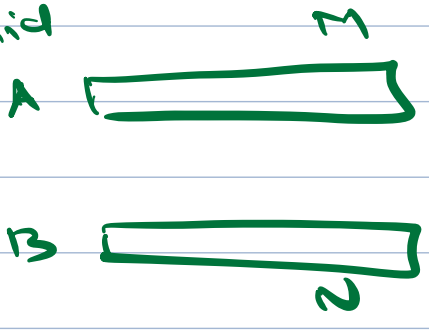
```
int total = M + N
int firsthalf = (M+N+1) / 2

l = 0        r = min(firsthalf, M)

while (l <= r) {

    int mid = (l+r) / 2

    int l1 = A[mid-1]
    int r1 = A[mid]

    int cntB = firsthalf - mid
    int l2 = B[cntB-1]
    int r2 = B[cntB]

    if (l2 > r1) {
        l = mid + 1        // right
    }
    else if (l1 > r2) {
        r = mid - 1        // left
    }
```

A: [2, 3]

B: [2, 7, 9, 1, 7, 8, 9]

Total = 9

firsthalf = (9+1)/2 = 5



```
else {
    if (total % 2 == 0) {
        median = max(l1, l2) + min(r1, r2) / 2
    }
    else {
        median = max(l1, l2)
    }
    return median
}
}
```

A : [1, 3, 4, 7, 10, | 12] (l_1, r_1)

B : [2, 3, 6, 15] | (l_2, r_2)

Default values of variables if no value assigned

$l_2 = -\infty$ $\quad r_1 = \infty$

$l_1 = -\infty$ $\quad r_2 = \infty$

Doubts

C → A × B
rows cols

A B
3 × 2

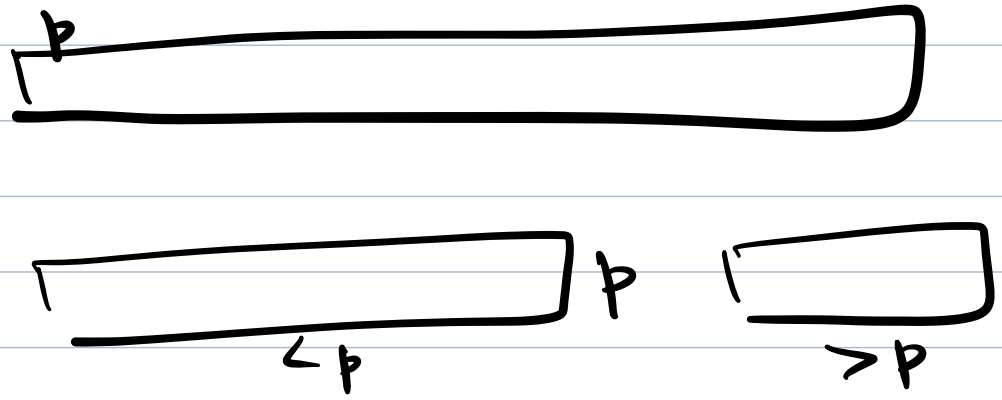
7, 3
2, 1
4, 9

[7, 1, 4]
↓ ↓
0 3 → 3

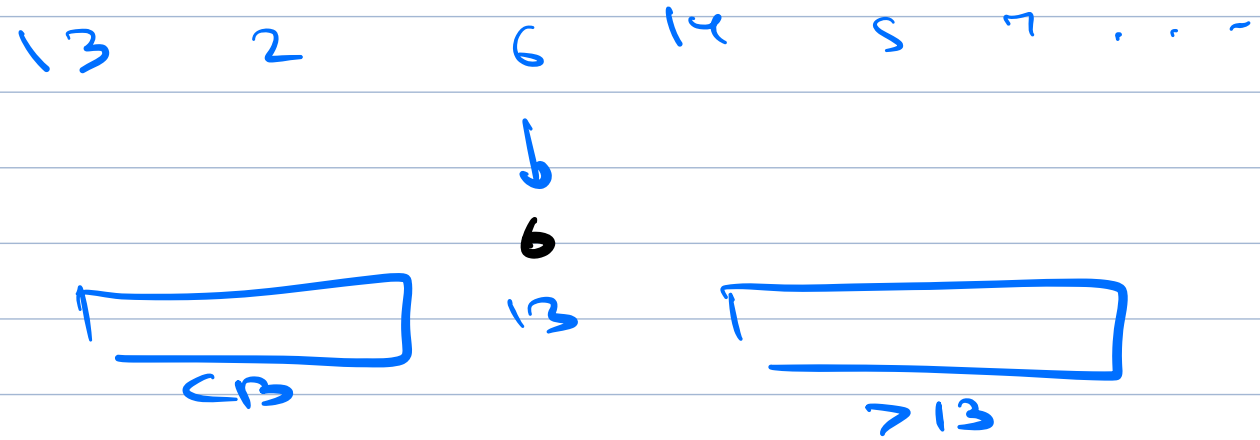
[3, 2, 4]
↓ ↓
1 2 → 1

[- , - - , -]
↓

→ 3 . . .
→ 1 . 2 . 7 9
. . . .

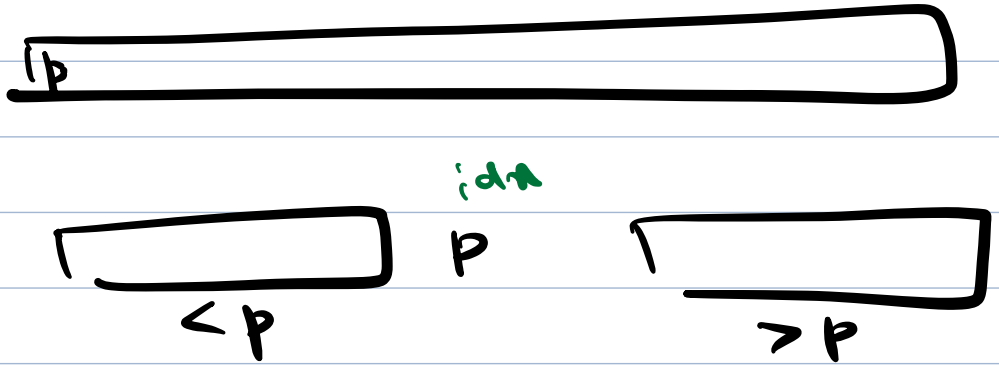


↳ smallest values



Given an array and B, return B smallest values

B



if idx ≥ B

qs (s, idx-1)

else

qs (idx+1, e)