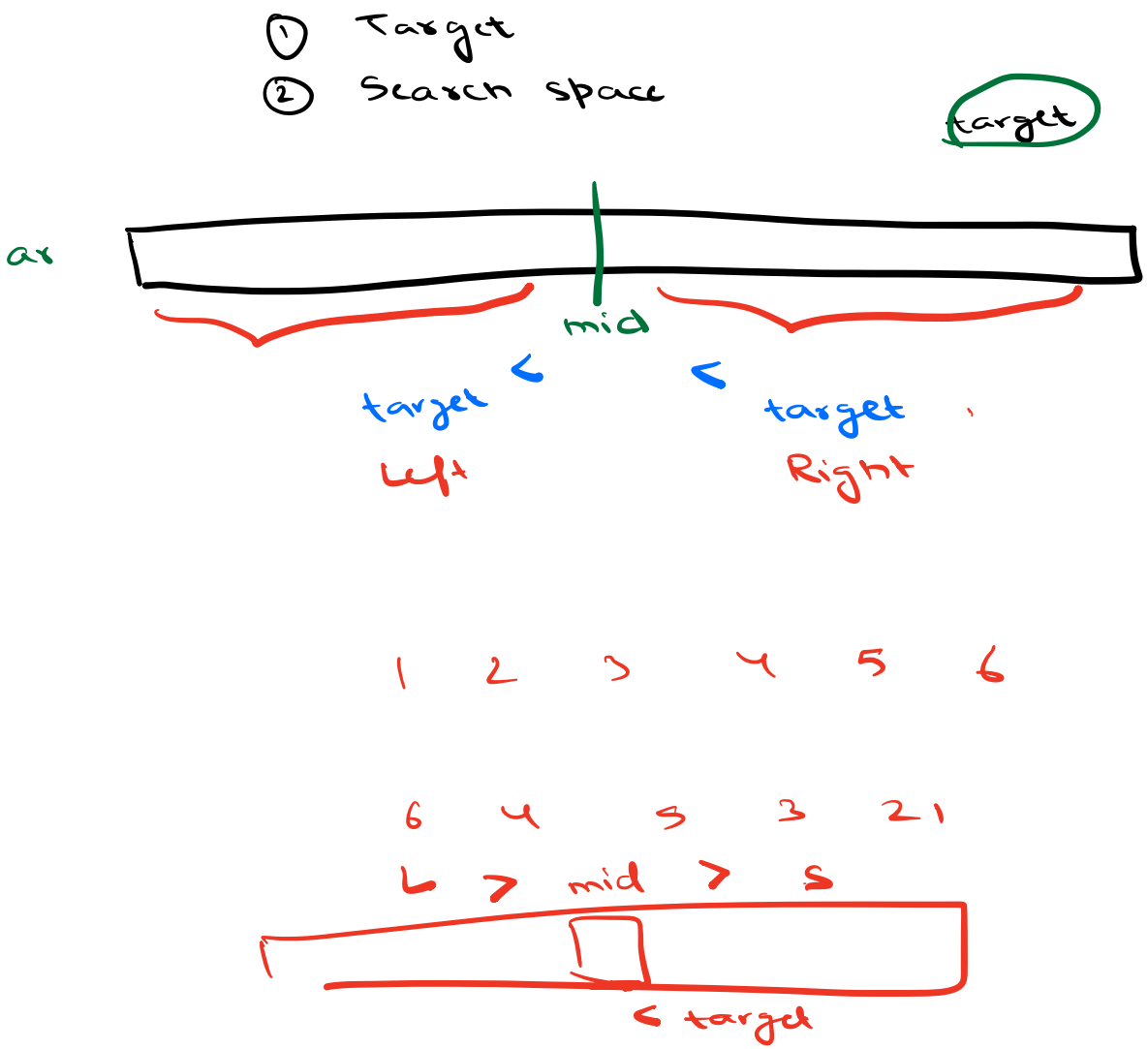
Searching in Rotated Sorted Array

Find square root of a number

An Magical Number

Median of 2 sorted Arrays



1. Find the target in a rotated sorted array (elements are distinct)

2 4 8 10 15

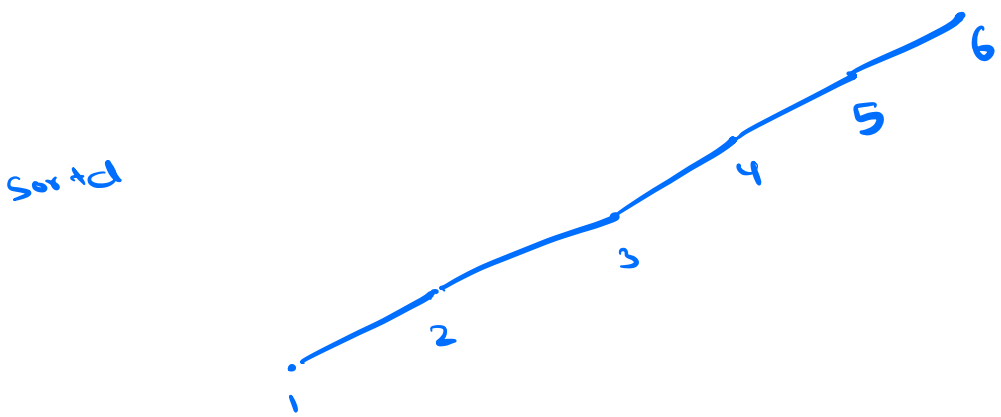
↓ rotated 2 times

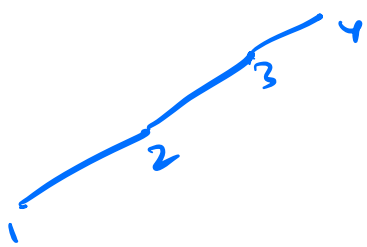
Input

0	1	2	3	4
10	15	2	4	8

Target = 2

Ans 2



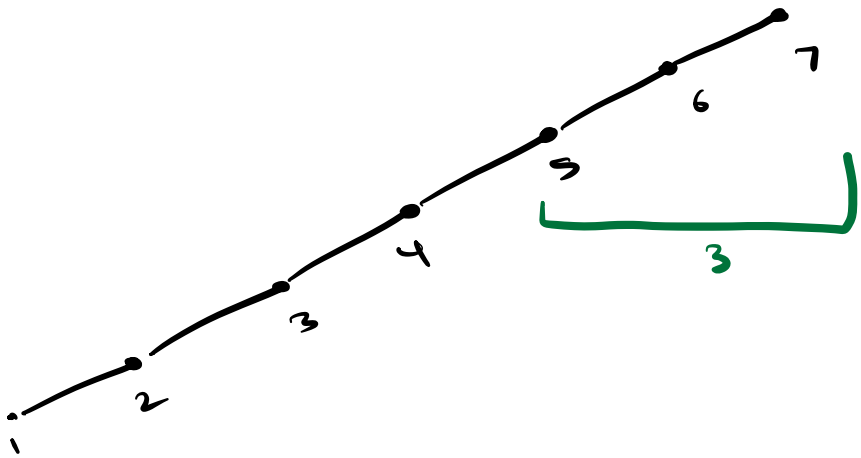


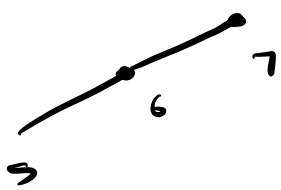
if (A[0] < A[N-1])

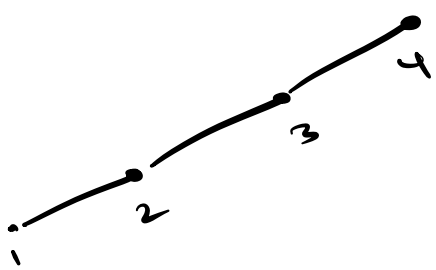
not rotated

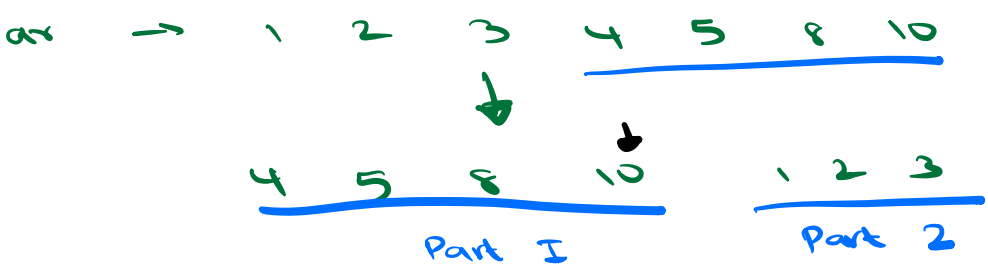
else

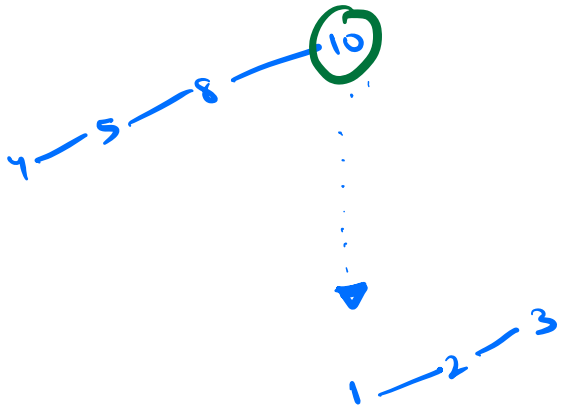
rotated











Find local maxima using BS

Then apply BS on part 1 and part 2

↓

TC : O(log n) → 3 log N

Approach 3 : Use BS only once

① 4 <= Part I

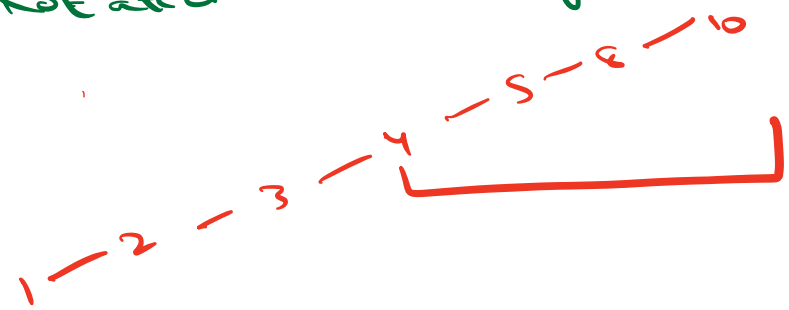
4 > Part 2

②

Part 1 → Part 2

Rotated

Original

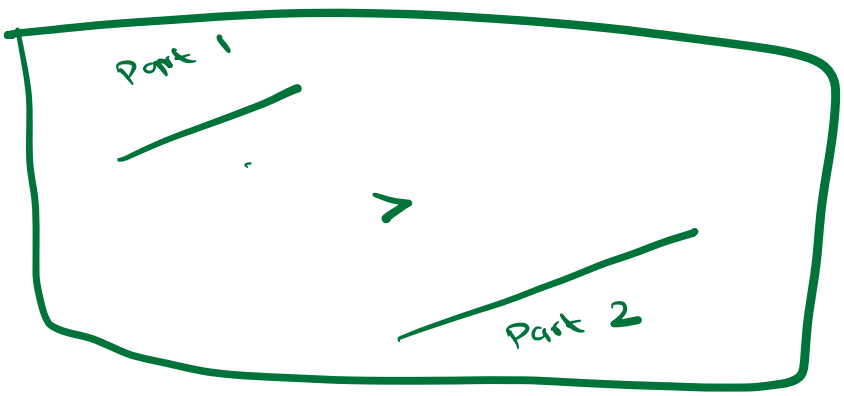


if (A[0] <= ele)

Part 1

else

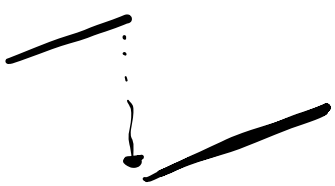
Part 2

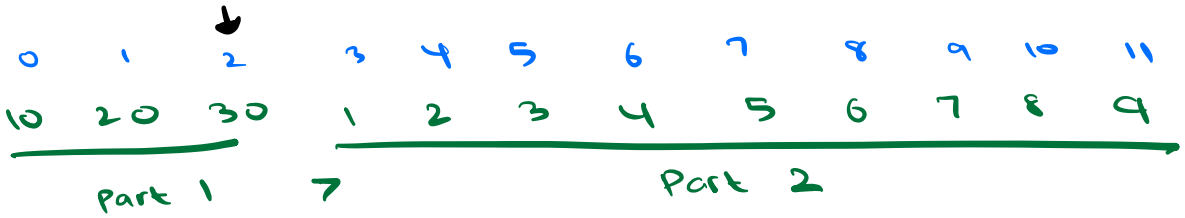


mid

target

Case 1	Mid is in part 1	Target is in part 2	Go right
Case 2	Mid in part 2	Target in part 1	Left
Case 3	Mid part 1	Target part 1	BS in part 1
Case 4	Mid part 2	Target part 2	BS in part 2





Target = 20

s	e	mid	Mid → Part	Target → Part	
0	11	5	Part 2	Part 1	Left
0	4	2	Part 1	Part 1	Left
0	1	0	Part 1	Part 1	Right
1	1	1	—	—	

A[mid] = A[target]
20 = = 20

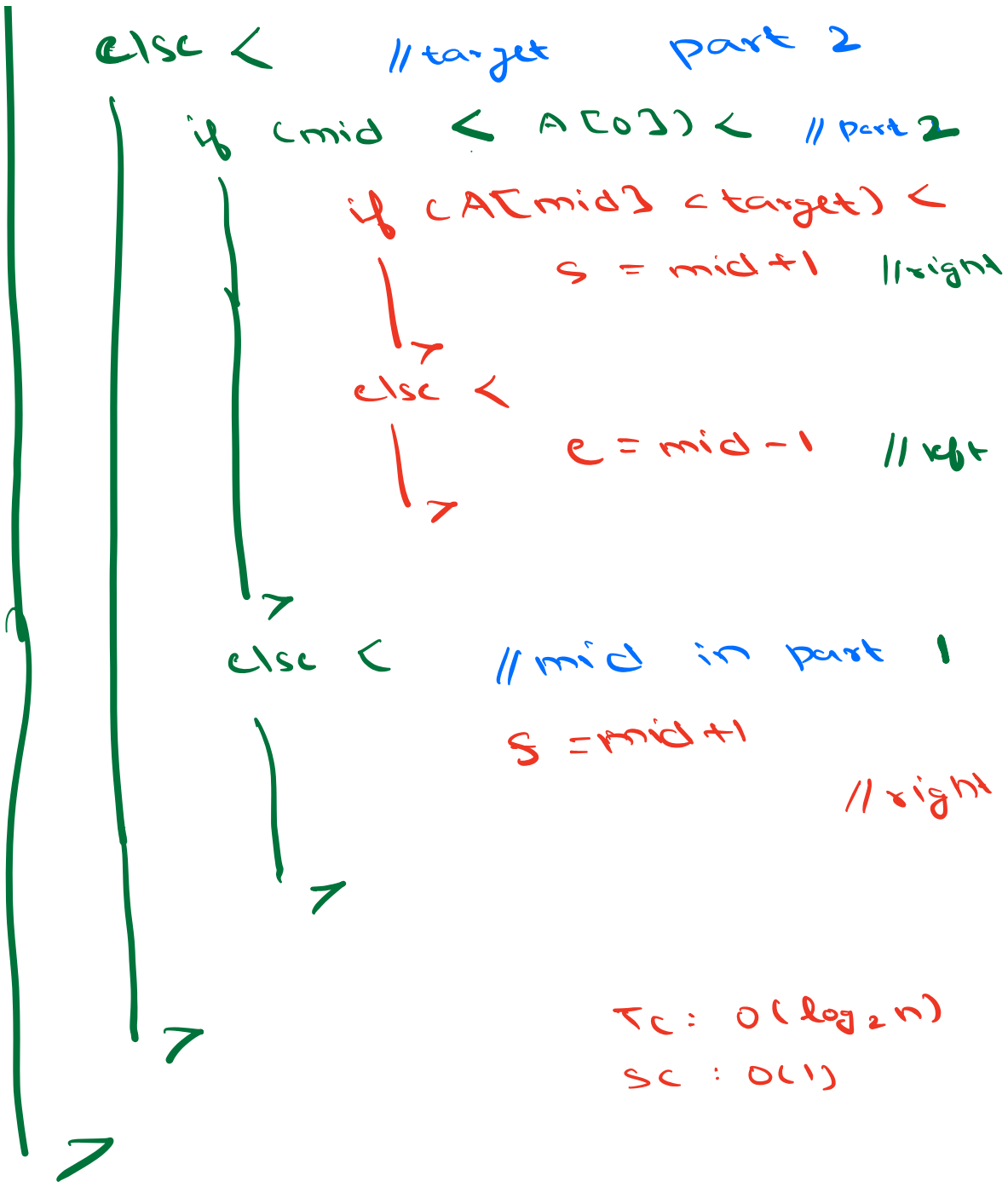
return mid
1

TC : log (search space)

```
s = 0, e = n-1
while (s <= e) {
    mid = (s + e) / 2

    if (A[mid] == target) {
        return mid
    }

    if (target >= A[0]) {          // part 1
        if (mid >= A[0]) {         // part 1
            if (A[mid] < target) {
                s = mid + 1        // right
            }
            else {
                e = mid - 1        // left
            }
        }
        else {                     // mid in part 2
            e = mid - 1            // left
        }
    }
}
```



TC: $O(\log_2 n)$

SC: $O(1)$

2. Given a positive no. N, find square root of N.

floor (square root (N))

N	Ans
25	5
20	4
10	3

$N \rightarrow x$

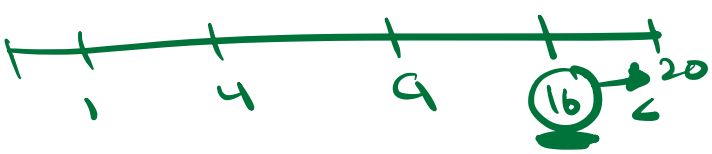
$x^2 = N$

	N	Ans
min →	1	1

```
i = 1
while (i*i <= N) {
    ans = i
    i++
}
```

i	N
1	25
2	
3	
4	
5	

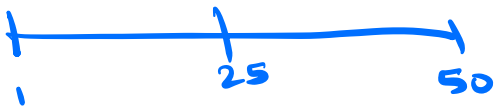
TC : $O(\sqrt{N})$ SC : $O(1)$



i	20
	$1^2 < 20$ ✓
	$2^2 < 20$ ✓
	$3^2 < 20$ ✓
	$4^2 < 20$ ✓
ans =	→ $5^2 > 20$

ans = 47

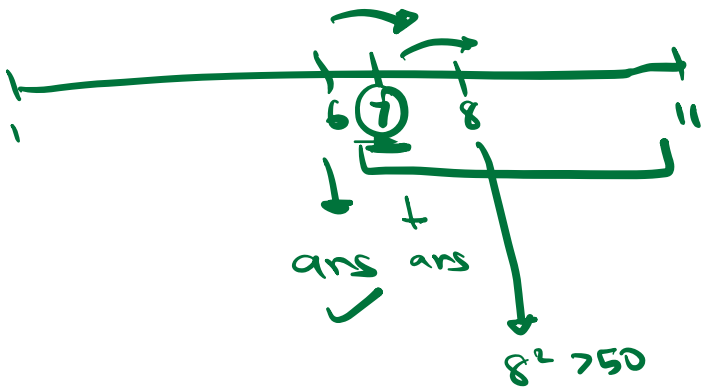
N = 50



s	e	mid		
1	50	25	25 * 25 > 50	left
1	24	12	12 * 12 > 50	left
1	11	6	6 * 6 < 50	right
			ans = 6	
7	11	9	9 * 9 > 50	left
7	8	7	7 * 7 < 50	right
			ans = 7	
8	8	8	8 * 8 > 50	left
9	7	break		

TC : O(log₂ N)

SC : O(1)



50

$x^2 < 50$

$6^2 < 50$

$7^2 < 50$

```
if (N == 0 || N == 1)
    return N

s = 1 , e = N , ans = 0
while (s <= e) {
    mid = (s + e) / 2

    if (mid * mid == N)
        return mid
    else if (mid * mid < N) {
        ans = mid
        s = mid + 1        // right
    }
    else {                 // mid * mid > N
        e = mid - 1        // left
    }
}
return ans
```

10:33

Median → Middle element in sorted data

4 5 10 13 17

Median = 10

4 5 10 13 17 20

Median = $\frac{10+13}{2} = \frac{23}{2} = 11.5$

3. Median of 2 sorted arrays

A → 1, 4, 5
B → 2, 3

ans = 3

A → 1, 2, 3
B → 4

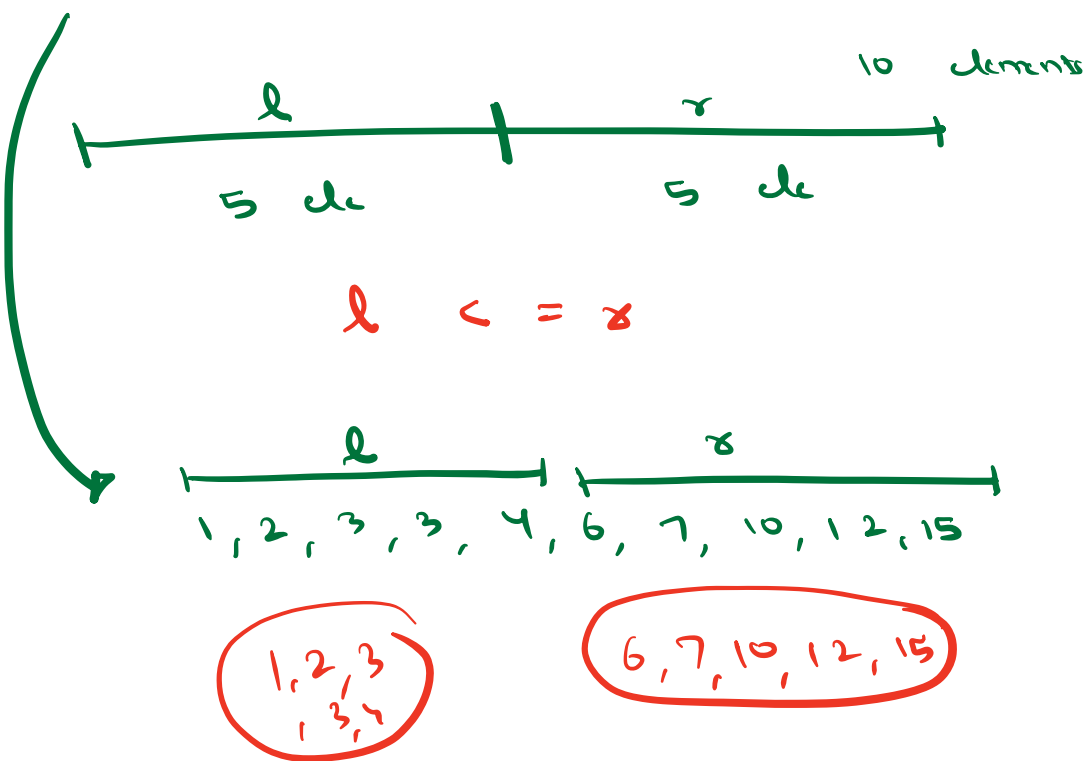
ans = 2.5

3 1 2
↓
1 2 3

Median → 2

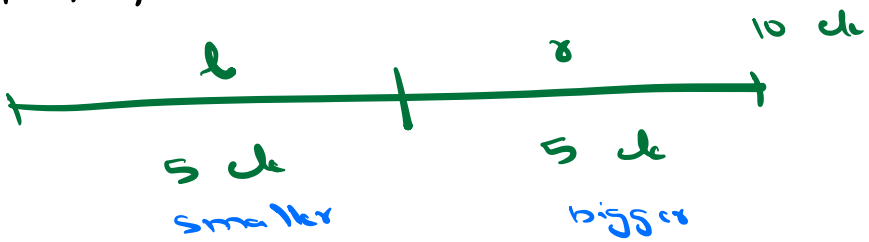
A → 1, 3, 4, 7, 10, 12
B → 2, 3, 6, 15

] 10 elements

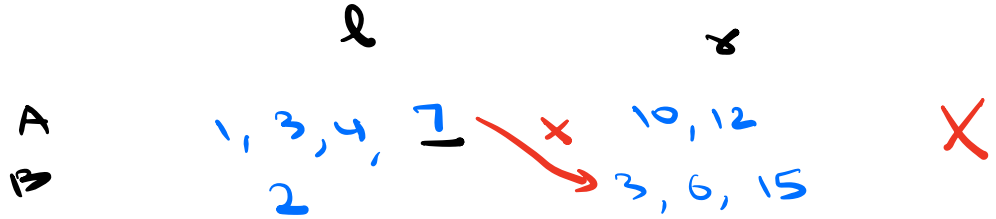


A → 1, 3, 4, 7, 10, 12

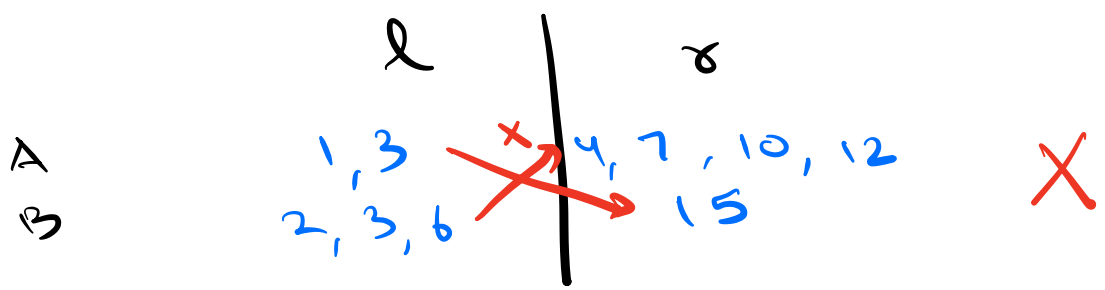
B → 2, 3, 6, 15



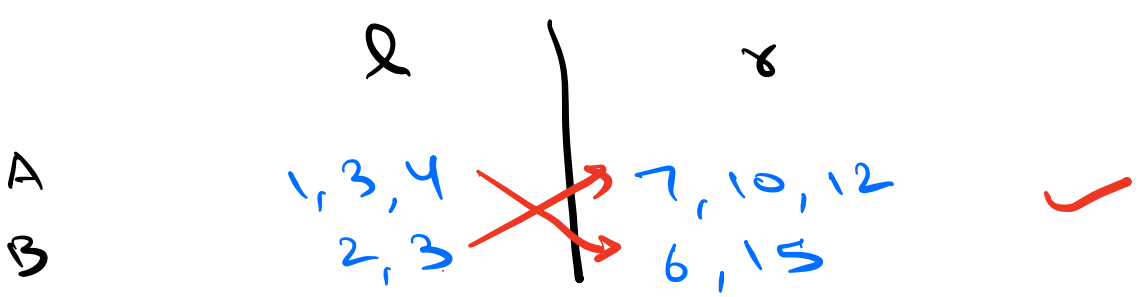
Case 1 : 4 elements from A

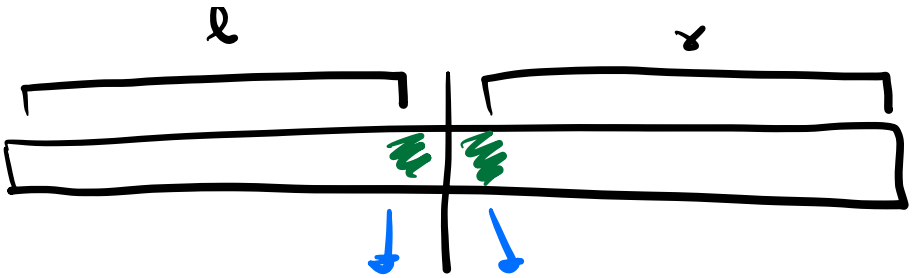


Case 2 : 2 de from A



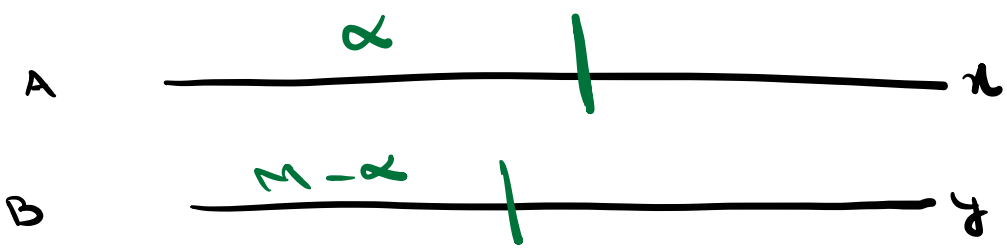
Case 3 : 3 de from A

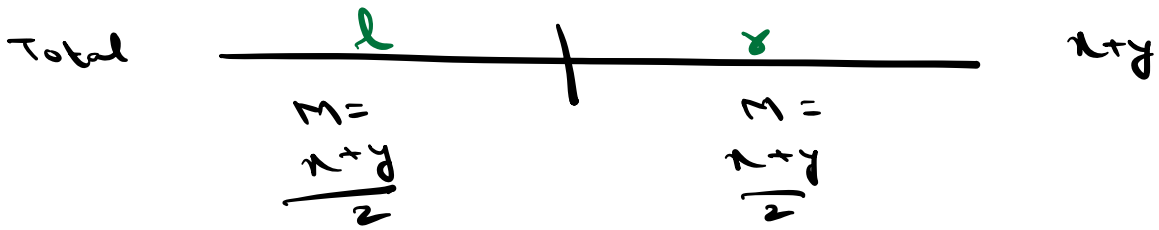




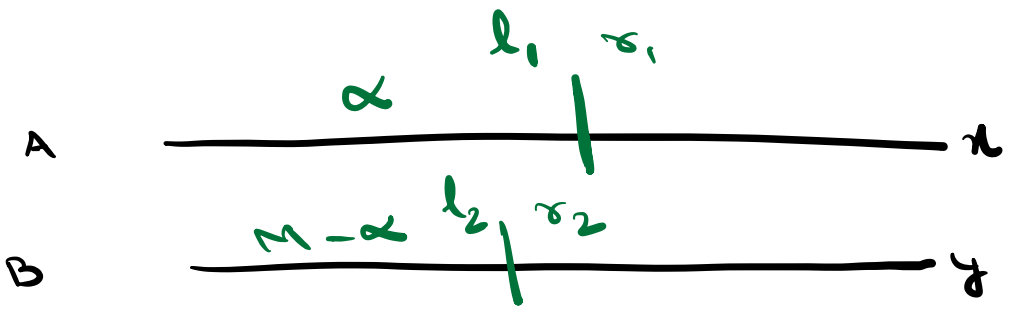
4 , 6

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



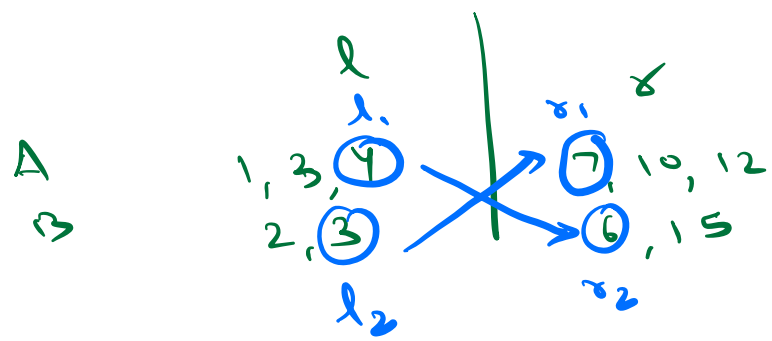


	l	r
A →	α	x − α
B →	M − α	y − (M − α)



A → 1, 3, 4, 7, 10, 12

B → 2, 3, 6, 15



$l_1 \rightarrow r_2$

$l_2 \rightarrow r_1$

$l <= r$

Check ($l_1 <= r_2$ and $l_2 <= r_1$)

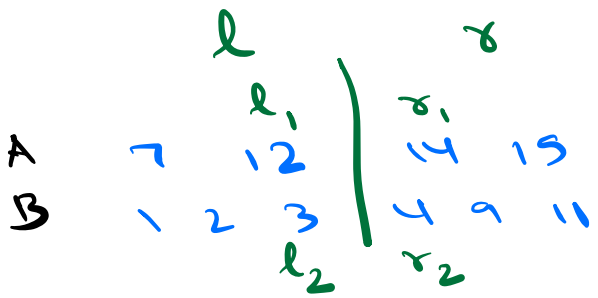
Median (Even) $\dfrac{\max(l_1, l_2) + \min(r_1, r_2)}{2}$

A → 7 12 14 15

B → 1 2 3 4 9 11] 10 elements

BS → no. of elements that you should pick from A to put on l half

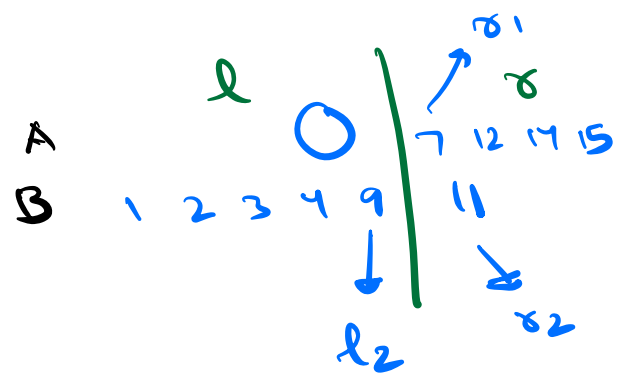
s = 0, e = 4, mid = 2



$l_1 <= r_2$

X $12 <= 4$ → Left $l_1 > r_2$

s = 0, e = 1, mid = 0



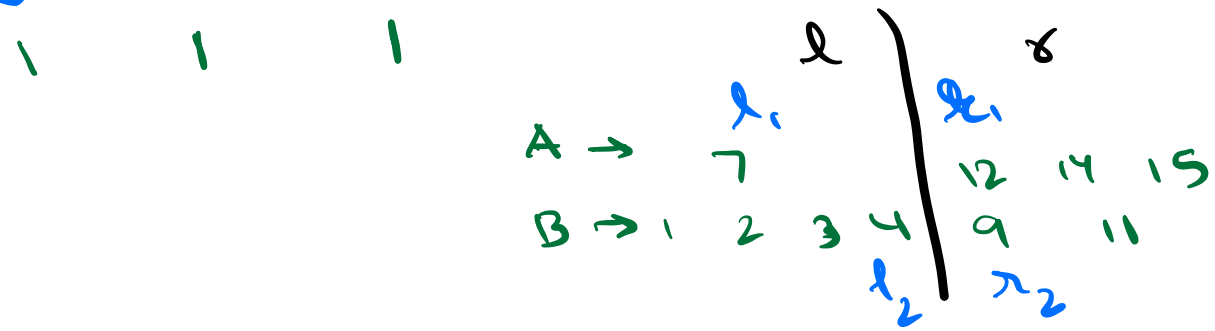
$l_1 <= r_2$ ✓

$l_1 = 0$

$l_2 <= r_1$

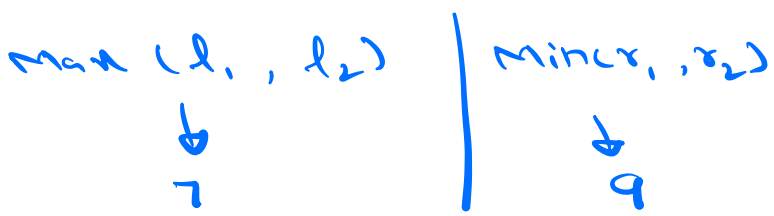
$9 <= 7$ ✗

right $l_2 > r_1$

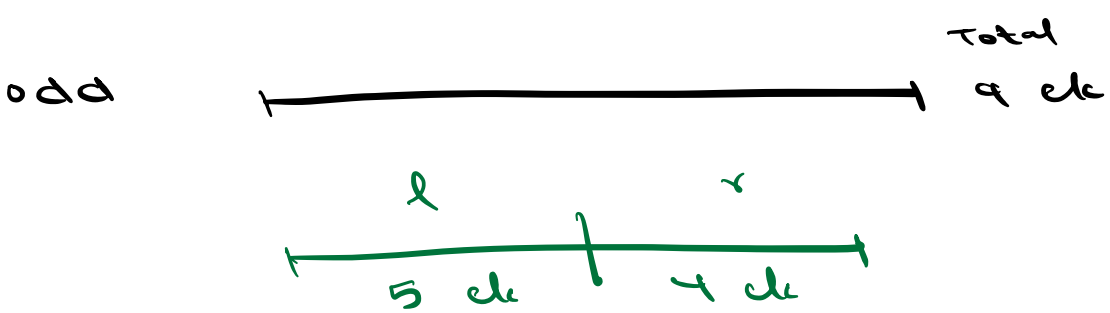


$l_1 <= r_2$ and $l_2 <= r_1$

$7 <= 9$ ✓ $4 <= 12$ ✓



Median = $\frac{7+9}{2} = 8$



Median = max(l_1, l_2)

smaller ar larger ar

```
findMedian( int[] A, int[] B ) {
    if (B.size < A.size) {
        return findMedian(B, A)
    }

    int m = A.size
    int n = B.size
    s = 0, e = m
    lhalfcnt = (n + m + 1)/2

    while (s <= e) {
        mid = (s + e)/2
```

↓ no. of el picked from A for left half

```
        cntA = mid
        cntB = lhalfcnt - cntA
        l1 = A[cntA - 1]
        l2 = B[cntB - 1]

        r1 = cntA == m ? ∞ : A[cntA]
        r2 = cntB == n ? ∞ : B[cntB]
```

-∞

-∞

no. of ele are even

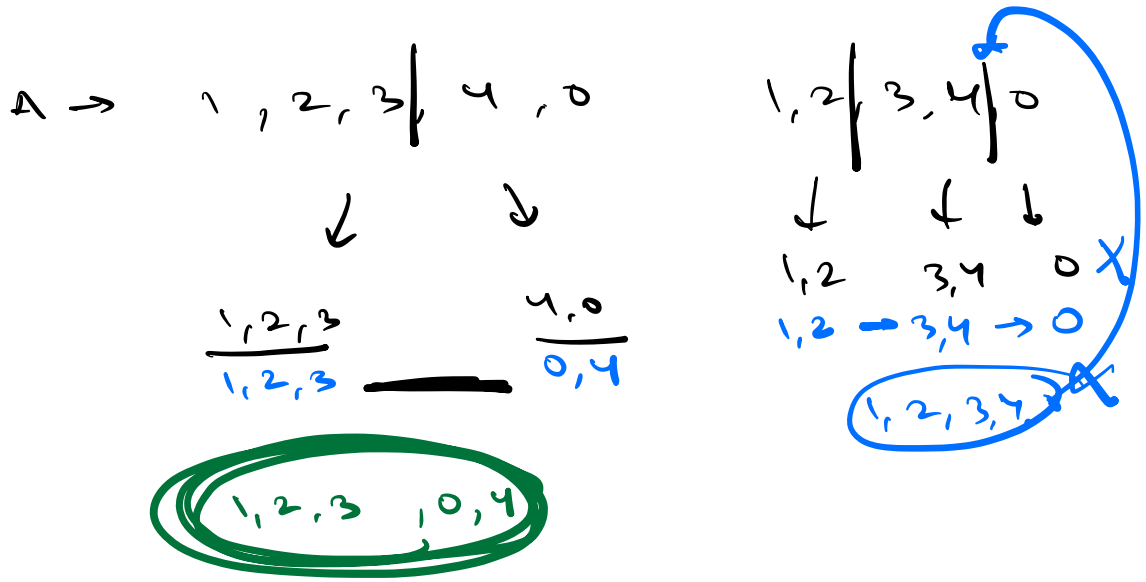

```
if (l1 <= r2 && l2 <= r1) {
    if (m + n % 2 == 0) {
        return max(l1, l2) + min(r1, r2) / 2;
    } else {
        return max(l1, l2);
    }
} else if (l1 > r2) {
    e = mid - 1    // left
} else {           // l2 > r1
    s = mid + 1    // right
}
}
```

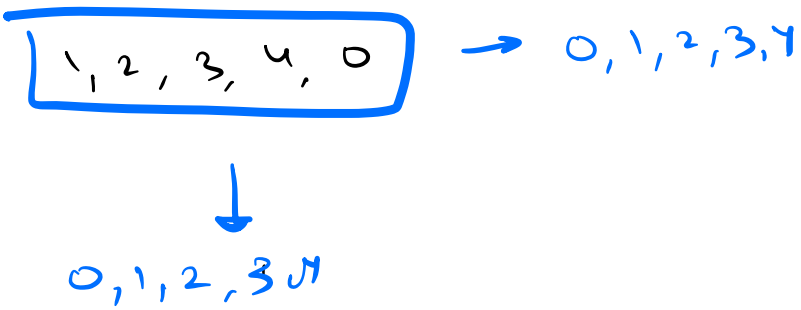
l_1 = cntA - 1 < 0 ? $-\infty$: A[cntA-1]

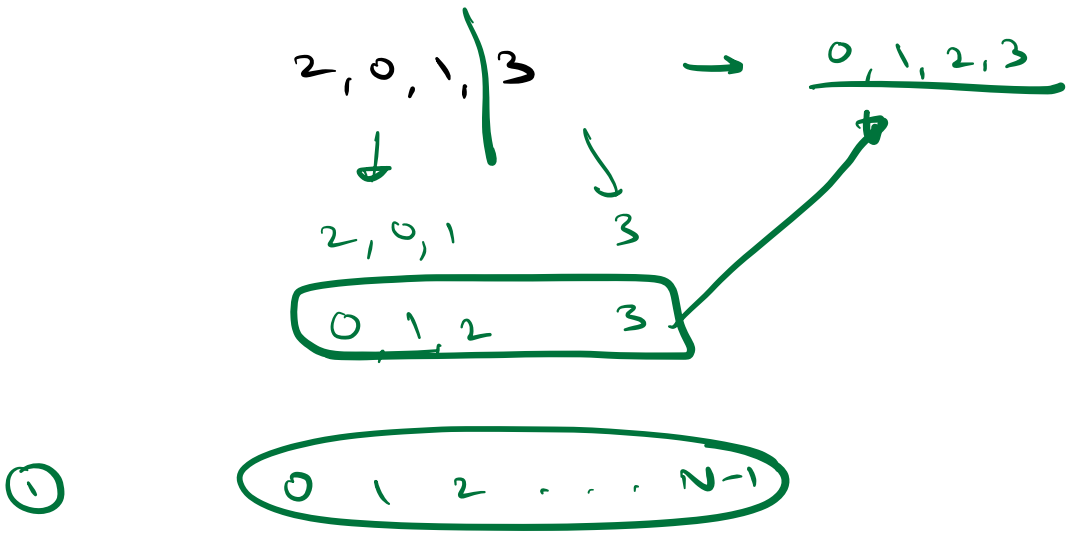
TC : O(log(min(N, M)))

SC : O(1)

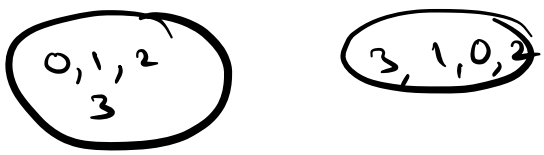
Doubts







A → 4



A → 4