# Lecture : Binary Search 2

## Agenda

**Qu1** Given a rotated sorted array, find an el in it.

input

|  |  |  |  |  |  | rotation |
|---|---|---|---|---|---|---|
| 2 | 4 | 8 | 10 | 15 | : Sorted array | 0 |
| 15 | 2 | 4 | 8 | 10 |  | 1 |
| 10 | 15 | 2 | 4 | 8 |  | 2 |
| 8 | 10 | 15 | 2 | 4 |  | 3 |
| 4 | 8 | 10 | 15 | 2 |  | 4 |
| 2 | 4 | 8 | 10 | 15 |  | 5 |

**Example:**

$$\begin{array}{lcccccl} & 0 & 1 & 2 & 3 & 4 & \\ \text{arr: } [ & 4 & 8 & 10 & 15 & 2 & ] \end{array} , \quad k = 15 \quad ans = 3$$

og: [ 2 4 8 10 15 ]

**Approach1** Linear search.

TC: O(n)

SC: O(1)

**Approach2:**

if array is sorted —

apply binary search

else —

do something else

og: [ 1 2 3 4 5 8 10 ]

$$\text{arr[] = } [\underbrace{4 \quad 5 \quad 8 \quad 10}_{\text{part1}} \quad \underbrace{1 \quad 2 \quad 3}_{\text{part 2}}]$$

10 → Peak el

**Challenge:** find the idx of peak element.

**Approach1** Traversal. Linear

TC: O(n)

SC: O(1)

**Approach2**

$$arr[] = [\underbrace{4 \quad 5 \quad 8 \quad 10}_{part1} \quad \underbrace{1 \quad 2 \quad 3}_{part2}]$$

all el (part1) > all el (part2)

Ref: 0th el is helping me doing the partition

all part1 els > arr[0]

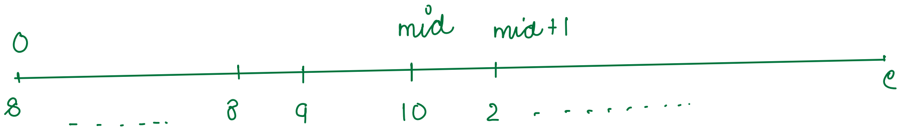
" part2 " < arr[0]

Target: peak el | max element

Search space: arr

**conditions**

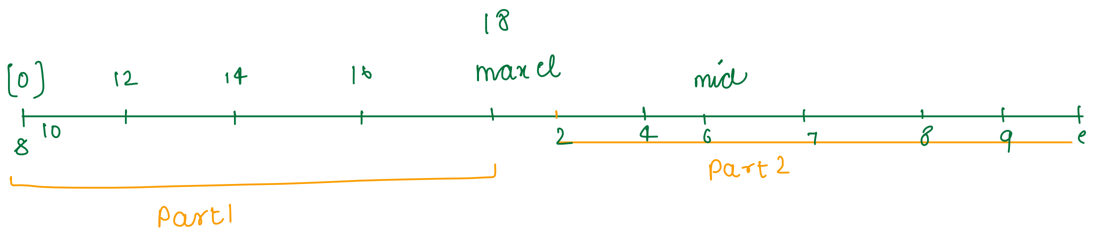
1.



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

return mid;

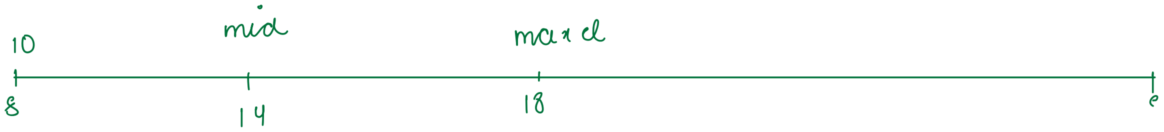
2.



if ( arr[mid] < arr[0]) — Part 2

e = mid -1 // left.

3.)



if ( arr[mid] > arr[0]) Part1

s = mid+1 // right

Dry run:

```
           0    1    2    3   4   5   6   7   8   9   10   11
arr[] = [ 10   20   30    1   2   3   4   5   6   7   8    9 ]
```

| s | e | mid | |
|---|---|---|---|
| 0 | 11 | 5 | arr[5] < arr[0]<br>e = mid-1 |
| 0 | 4 | 2 | arr[2] > arr[3] return mid = 2. |

arr[] = [ $\overset{0}{60}$ $\overset{1}{70}$ $\overset{2}{80}$ $\overset{3}{90}$ $\overset{4}{100}$ $\overset{5}{10}$ $\overset{6}{20}$ $\overset{7}{30}$ ]

| s | e | mid | |
|---|---|---|---|
| 0 | 7 | 3 | arr[3] > arr[0]<br>s = mid+1 |
| 4 | 7 | 5 | arr[5] < arr[0]<br>e = mid-1 |
| 4 | 4 | 4 | arr[4] > arr[5]<br>return 4. |

arr[] = [ $\overset{0}{4}$ $\overset{1}{4}$ $\overset{2}{4}$ $\overset{3}{4}$ $\overset{4}{4}$ $\overset{5}{5}$ $\overset{6}{4}$ $\overset{7}{4}$ ]

| s | e | mid | |
|---|---|---|---|
| 0 | 7 | 3 | arr[3] not greater arr[0]<br>right |

<u>Hint:</u> What if I encounter a 4 on part 2 ?

How to make sure to move to right / left

Tweak. <u>H/W</u>

```
int findLargestElementIdx( int [] arr) {
        s=0;   e = arr.length-1;
        while( s<=e) {
            mid = s + (e-s)/2 ;

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

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

```
          0    1    2    3   4   5   6   7   8   9   10   11
arr[] = [ 10   20   30   1   2   3   4   5   6   7   8    9 ]
               beak-idx
```

```
      int  searchInRotatedSortedArray (int[] arr, int k) {
O(logn) —— int  beak-idx = findLargestElementIdx (arr);
O(logn) — int ans  = binarysearch (arr, 0, beak-idx);
            if ( ans != -1) {
                return ans;
            }
     O(logn.——return binarysearch (arr, beak-idx+1, arr.length-1);
}
```

TC: $O(3\log_2 n) \simeq O(\log_2 n)$

SC: $O(1)$

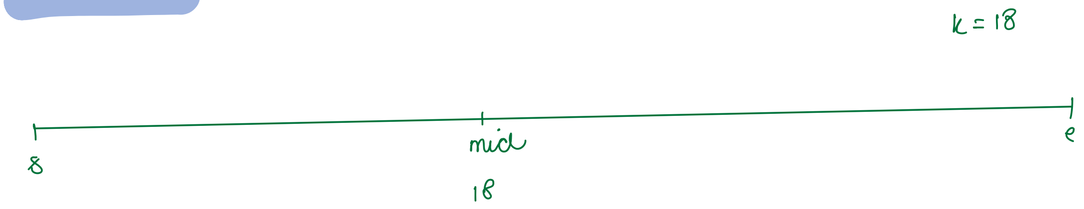
Approach3: Do in one binary search.

Target = k

search space = array

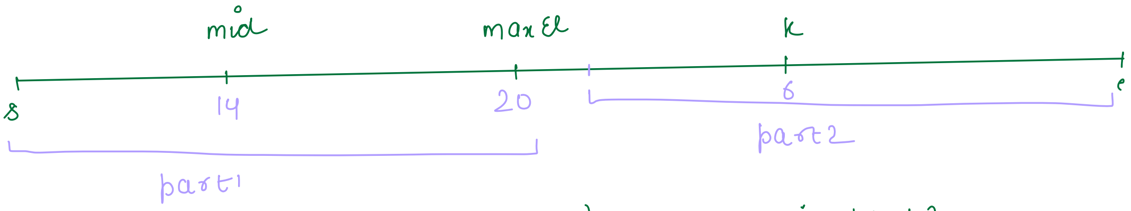
conditions

1.

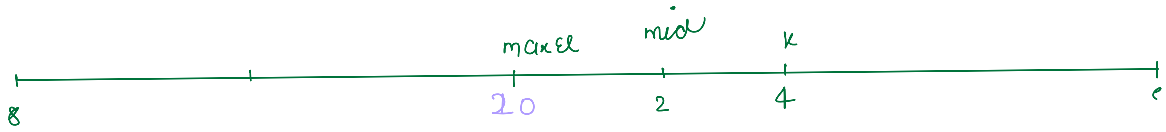


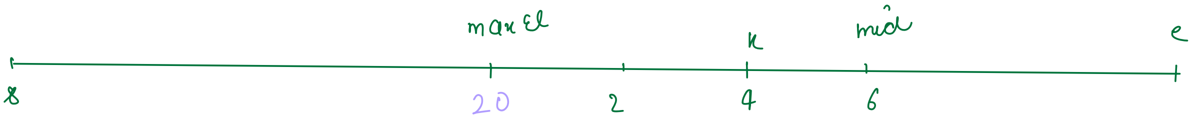
```
if( arr[mid] == k ) { return mid};
```

2.)

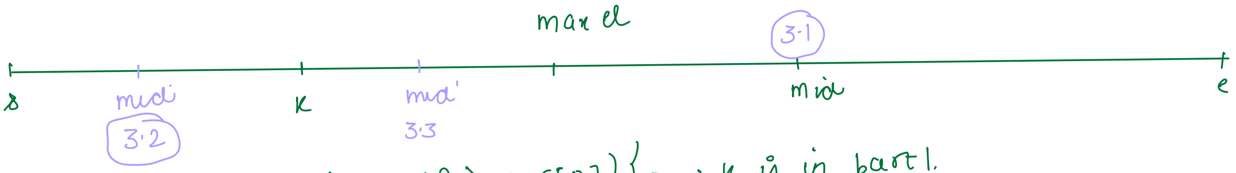


```
if( k < arr[0])  →  k is in part2

    if( arr[mid] > arr[0])  — mid is in part 1.

        s = mid+1;

    } else {  — mid is in part 2

        if( arr[mid] > k){
            e = mid-1;
        } else {

            s = mid+1;
        }
}
```





3)



```
if( arr[k] > arr[0]) {  → k is in part1.

3.1    if ( arr[mid] < arr[0])  → mid is in part 2
              e = mid-1;
       } else{        → mid is in part1.

3.2         if ( arr[mid] < k ) {
                 s = mid+1;
3.3         } else{
                 e = mid-1
            }
}
```

Dry run:

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| arr[] = [ | 10 | 20 | 30 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ] |

k = 20

| s | e | mid |  |
|---|---|---|---|
| 0 | 11 | 5 | k >arr[0] — k is in part1.<br>arr[mid] <arr[0]— mid is in part 2<br>left ; e= mid-1. |
| 0 | 4 | 2 | k is in part1<br>arr[mid] > arr[0] — mid is in part1.<br>arr[mid] > k<br>e = mid-1 |
| 0 | 1 | 0 | k is in part1<br>arr[mid] >= arr[0] → mid is part 1.<br>arr[mid] <k. ⟶ s = mid+1 |
| 1 | 1 | 1 | arr(mid) == k<br>return mid; |

```
int searchInRotatedSortedArray(arr[], int k){
    s=0; e= arr.length-1.
    while(s<=e){
        mid = s + ((e-s)/2);

        if(arr[mid]==k){
            return mid;
        }
        if( k < arr[0])  ⟶  k is in part2
            if ( arr[mid] > arr[0])  — mid is in part 1.
                s = mid+1;
            } else {  — mid is in part 2
                if( arr[mid] > k){
                    e = mid-1;
                } else {
                    s = mid+1;
                }
            }
        } else {
3.1         if ( arr[mid] < arr[0])  ⟶  mid is in part 2
                e = mid-1;
            } else {  ⟶  mid is in part 1.
3.2             if ( arr[mid] < k) {
                    s = mid+1;
3.3             } else {
                    e = mid-1
                }
            }
        }
    return -1;
}
```

TC: $O(\log_2 n)$

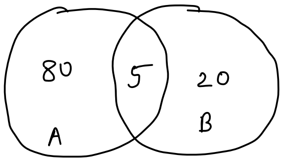
SC: $O(1)$

Break: 8:40 AM

lcm * hcf = a*b.

$$\boxed{lcm(a,b) = \frac{a*b}{gcd(a,b)}}$$

venn diagram



A: students who play cricket: 80

B: " " " football: 20

students who play both cricket $\overset{\cap}{\&}$ football : 5

" " " " " $\underset{\cup}{\text{(or)}}$ " : ?

$A \cup B = A + B - A \cap B$

$= 80 + 20 - 5 = 95$

Qu: Given A and B, find no of multiples of A from [1-B]

A B

3 20 : 3 6 9 12 15 18 :- 20/3

4 30 : 4 8 12 16 20 24 28 : 30/4

7 35 : 7 14 21 28 35 : 35/5

Qu  Given A, B and C. find no of multiples of B or C from [1 - A]

| B | C | A |
|---|---|---|
| 3 | 5 | 35 |

Multiples of 3: 3  6  9  12  15  18  21  24  27  30  33 : 35/3

Multiples of 5: 5  10  15  20  25  30  35 : 35/5

Multiples of 3 or 5 : $\frac{35}{3} + \frac{35}{5}$ — common multiples of 3 & 5

$$\Rightarrow \frac{35}{3} + \frac{35}{5} - \frac{35}{\text{Lcm}(3,5)}$$

$$\Rightarrow \frac{35}{3} + \frac{35}{5} - \frac{35}{15}$$

$$\Rightarrow 11 + 7 - 2 = 16$$

Qu: A th Magical number

Given A, B and C. find Ath magical number

Note: A is said to be magical if it is divisible by B or C.

Ex. b c a

2 3 8 :- 2 3 4 6 8 9 10 12

↑ 8 th magical no.

4 6 5 :-

| 4 | 6 | 8 | 12 | 16 |
|---|---|---|---|---|
| 1st magical no | 2nd | 3rd | 4th | 5th |

5 9 10 :-

| 5 | 9 | 10 | 15 | 18 | 20 | 25 | 27 | 30 | 35 |
|---|---|---|---|---|---|---|---|---|---|
| 1st | 2nd | 3rd | 4th | 5th | 6th | 7th | 8th | 9th | 10th |

| 36 | 40 | 45 | 50. |
|---|---|---|---|
| 11th | 12th | 13th | 14th |

Brute force:

min possible ans :- min(b, c).

max possible ans :- min(b, c) * a.

b = 2

c = 100

a = 20

2 4 6 8 - - - ------ 40

↑ 20th magical no

```
int   findMagicalNo( int a, int b,  int c) {

        int  cnt =0;

        int  min  =  min (b,c);
        int  max =  min(b,c) * a;

        for( i= min; i<= max; i++){
                if ( i % b==0||  i % c==0){
                        cnt ++;
                }
                if (cnt == a) {
                    return i;
                }
        }
}
```

TC: O(max) ≃ O(a*min(b,c))

SC: O(1)

Approach2:

Target = Ath magical no

Search space:- [ min(b,c) to min(b,c) * a ]

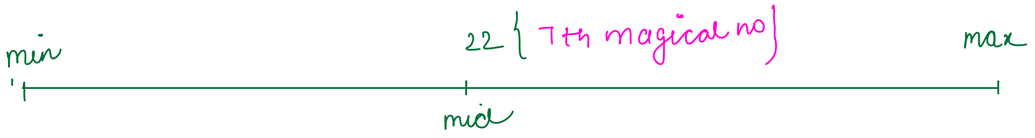
Condition

Idea:

| b | c | a |
|---|---|---|
| 4 | 6 | 10 |

min = 4

max = 40 [ 4 *10 ]



↑

Check 22 is which magical no?

$$\frac{22}{4} + \frac{22}{6} - \frac{22}{lcm(4,6)}$$

$$\frac{22}{4} + \frac{22}{6} - \frac{22}{12}$$

$$5 + 3 - 1 = \boxed{7}$$

Ex2:

| B | C | A |
|---|---|---|
| 6 | 9 | 4 |

→ 6 9 12 18 24 27 - - -

Random mids

18 :- $\frac{18}{6} + \frac{18}{9} - \frac{18}{lcm(9,6)}$

$= \frac{18}{6} + \frac{18}{9} - \frac{18}{18} = 4.$

$\boxed{19}$ :- $\frac{19}{6} + \frac{19}{9} - \frac{19}{18} =$ 3 + 2 − 1 = 4.

## Tracing

| B | C | A |
|---|---|---|
| 5 | 7 | 4 |

min = 5

max = 20

| s | e | mid | |
|---|---|---|---|
| 5 | 20 | 12 | $\frac{12}{5} + \frac{12}{7} - \frac{12}{35} = 3$ |
| | | | s = mid+1. |
| 13 | 20 | 16 | $\frac{16}{5} + \frac{16}{7} - \frac{16}{35} = 5$ |
| | | | e = mid-1 |
| 13 | 15 | 14 | $\frac{14}{5} + \frac{14}{7} - \frac{14}{35} = 4$ [ ans = 4 ] |
| | | | e = mid-1 |
| 13 | 13 | 13 | $\frac{13}{5} + \frac{13}{7} - \frac{13}{35} = 3$ |
| | | | s = mid+1 |
| 14 | 13 | break | |

```
int  AthMagicalNo( int a. int b, int c){
    int  s  =  min (b,c);
    int  e  =  min (b,c) * a;
    int lcm = $\frac{a*b}{gcd(a,b)}$ ;      ans = 0;

    while( s <= e){
        mid = s + $\frac{(e-s)}{2}$;

        int magical = $\frac{mid}{b} + \frac{mid}{c} - \frac{mid}{lcm}$.

        if (magical < a){
            s = mid+1;
        }  else if ( magical > a){
            e = mid-1;
        }  else {
            ans = mid;
            e = mid-1;
        }
    }
    return ans;
}
```

TC: $O(\log_2(\min(b,c) * a))$

SC: $O(1)$

Qu: sqrt of a number.

Eg: sqrt(25) = 5

sqrt(26) = 5

sqrt(24) = 4

Approach1:

```
int sqrt(n) {
    i = 1,  ans = 1;

    while( i*i <= n) {
        ans = i;
        i++;
    }
    return ans;
}
```

50

| i | ans |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | $\boxed{7}$ Ans |
| 8 | |

Approach2:

sqrt(n)

min = 1

max = n.

n=100   sqrt[1 - 100]   ans=10

n=1   sqrt[1-1]   ans=1.

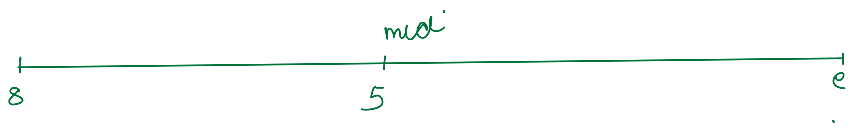
Target: floor sqrt(n)

Range / search space = [1-n]

condition

n=25

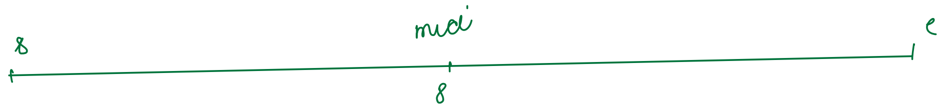
1.



mid * mid == n ? return mid.

n=25

2.

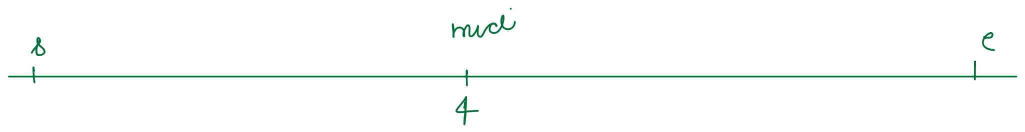


mid * mid > n.

e = mid-1

n=24

3.



mid * mid < n.

s = mid+1

ans = mid

n=50

| s | e | mid | | |
|---|---|---|---|---|
| 1 | 50 | 25 | 25*25 > 50 | e = mid-1 |
| 1 | 24 | 12 | 12*12 > 50 | e = mid-1 |
| 1 | 11 | 6 | 6*6 < 50 | ans = 6 , s = mid+1. |
| 7 | 11 | 9 | 9*9 > 50 | e = mid-1 |
| 7 | 8 | 7 | 7*7 < 50 | ans = 7 , s = mid+1 |
| 8 | 8 | 8 | 8*8 > 50 | e = mid-1 |
| 8 | 7 | break | | |

```
int sqrt ( int n) {
        s=1, e=n, ans;
        while ( s<=e){
            mid = s+((e-s)/2);
            if( mid * mid < n){
                s= mid+1;
                ans= mid;
            } else if ( mid*mid>n){
                e = mid -1;
            } else {
                return mid;
            }
        }
    return ans;
}
```

TC: $O(\log_2 n)$

SC: $O(1)$

Thankyou ☺