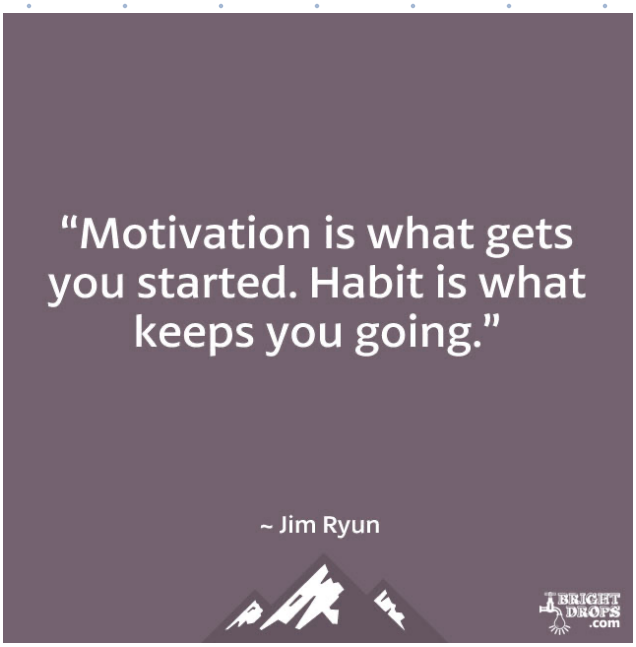
Binary Search 2



Good Evening

Today's content

01. sqrt(N)
02. Search in sorted rotated array
03. A* magical no.
04. Median of an array

01. Given a positive no. N, find sqrt(N)/

Given N, find greatest i such that $i * i \leq N$

floor(sqrt(n))

sqrt(25) ⇒ 5

sqrt(16) ⇒ 4

sqrt(52) ⇒ 7

sqrt(48) ⇒ 6

Idea 1 → Start iterating from 1 & keep updating your ans until $i*i \leq N$

TC: $O(\sqrt{n})$

SC: O(1)

N = 40

i	i*i ≤ N	ans
1	1*1 ≤ 40	1
2	2*2 ≤ 40	2
3	3*3 ≤ 40	3
4	4*4 ≤ 40	4
5	5*5 ≤ 40	5
6	6*6 ≤ 40	6
7	7*7 ≤ 40 → Break;	

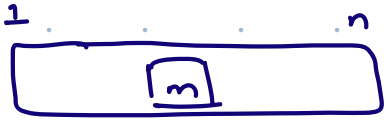
Ans = 6

Idea 2 Binary search

Target = $\sqrt{n}$ Search space ⇒ 1 to N

How to reduce search space?

Case 1



if (m * m == N) return m;

Case II

if (m*m > n) move left

Case III

if (m*m < n){

ans = m

"go to Right

}

N = 50

lo	hi	mid = (lo+hi)/2	compare mid*mid & N
1	50	25	25 * 25 > 50 : hi = m-1
1	24	12	12 * 12 > 50 : hi = m-1
1	11	6	6 * 6 < 50 ans = 6, lo = m+1

7	11	9	9*9 > 50 : hi = m-1
7	8	7	7*7 < 50 : ans = 7, lo = m+1
8	8	8	8*8 > 50 : hi = m-1
8	7	→ Break	

Ans = 7

```
ans = -1;
lo = 1      hi = N
while (lo ≤ hi) {
    int m = (lo+hi)/2;
    if (m*m == n) return m;
    else if (m*m > n) {
        hi = m-1;
    }
    else {
        ans = m;
        lo = m+1;
    }
}
return ans;
```

TC : O(log N)

SC : O(1)

* Sorted arr[] = {3, 9, 14, 16, 20, 24}:

Given K, can this K be present inside this array or not?

K = 17 → Yes, it is a possibility

K = 27 → No

Claim → If I have a sorted array, then I can figure out if an element is possibly present or not.

* **Rotated sorted Array**

A[]: {1 2 3 4 5}

K = 0 {1 2 3 4 5}

K = 1 {5 1 2 3 4}

K = 2 {4 5 1 2 3}

K = 3 {3 4 5 1 2}

K = 4 {2 3 4 5 1}

K = 5 {1 2 3 4 5}

Claim → If I have sorted rotated array, it'll always have atleast one half sorted based on a particular index

Search in sorted rotated Array

* Given an input ar[], formed by rotating a distinct sorted array right to left.

Search ele & return index if it is present else return -1.

ar[] =

10	11	12	13	17	20	23	25	26	1	3	5	6	8
0	1	2	3	4	5	6	7	8	9	10	11	12	13

K = 17

Ans = 4

01. Linear search if (A[i] == K)

TC : O(n)
SC : O(1)

02. Binary Search (TODO)

→ Find the peak ele idx of array -

→ Split your array in two subarray

0 to peak ele idx peak ele idx +1 to n-1

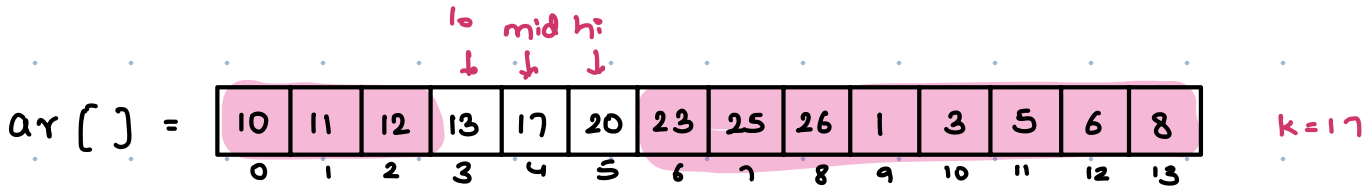
→ Figure out where K will lie based upon si & ei of subarray

→ Apply BS again in subarray which can possibly have this K.

03 Use binary Search

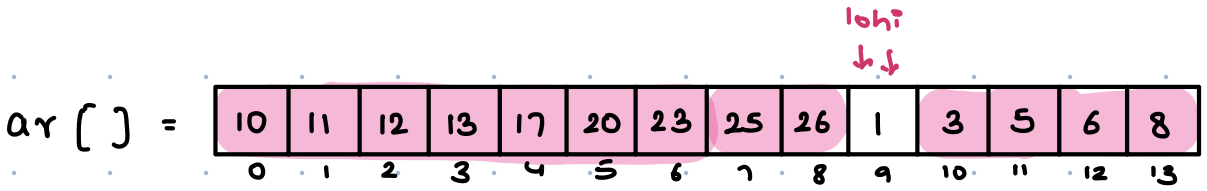
Target = k

Search space = 0 to n-1



lo	hi	mid	
0	13	6	: left half is sorted check if k is b/w 0 to mid, Yes : hi = m-1
0	5	2	: left half is sorted check if k is b/w 0 to mid, Not lo = m+1
3	5	4	⟶ return idx

m
↓



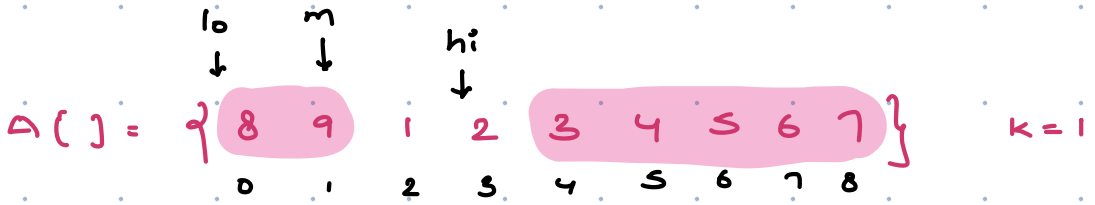
k = 1

lo	hi	mid	
0	13	6	: left half is sorted 10 > 3 && 3 < 23 → not lying in sorted subarr lo = m+1
7	13	10	: right half is sorted 1 < 5 && 1 < 8 → not lying in right half hi = m-1
7	9	8	: left half is sorted 1 < 25 && 1 < 26 → not lying in left half : lo = m+1
9	9	9	→ return 9

```
lo = 0          hi = n-1

while (lo ≤ hi) {
    int m = (lo + hi)/2 ;
    if (A[m] == K) return m;
    else if (A[lo] ≤ A[m]) {        { lo ... m is sorted }
        if (A[lo] ≤ K && K < A[m]) hi = m-1
        else lo = m+1;
    }
    else {        { m .... hi is sorted }
        if (A[m] < K && K ≤ A[hi]) lo = m+1
        else { hi = m-1; }
    }
}
return -1;
```

10:06 PM → 10:17 PM



* A^{th} magical no.

A number is magical if it is divisible by B or C

B = 2
C = 3
A = 8
} Find the 8th magical divisible by 2 or 3

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

Ans = 12

B = 3
C = 7
A = 4
}

1 2 3 4 5 6 7 8 9 → Ans

B = 3
C = 17
A = 2
}

1 2 3 4 5 6 Ans

Brute force Idea

→ Start iterating from 1 & check if it is divisible by B or C.

```
if ( i % B == 0 || i % C == 0 ) count ++;

if ( count == A ) return i ;
```

Max possible no. ⇒ A * min(B,C)

Search space = [1 ... A * min(B,C)]

min(B,C)

Target = A^th Magical no.

* B = 5, C = 3 } find the count of magical no. from 1 to 35

3 5 6 9 10 12 15 18 20 21 24 25 27 30 33 35

Mul of 3 in range [1-35] = $\frac{35}{3}$ = 11

Mul of 5 in range [1 35] = $\frac{35}{5}$ = 7

Mul of 3*5 in range [1-35] = $\frac{35}{15}$ = 2

} Ans = 11+7-2 = 16

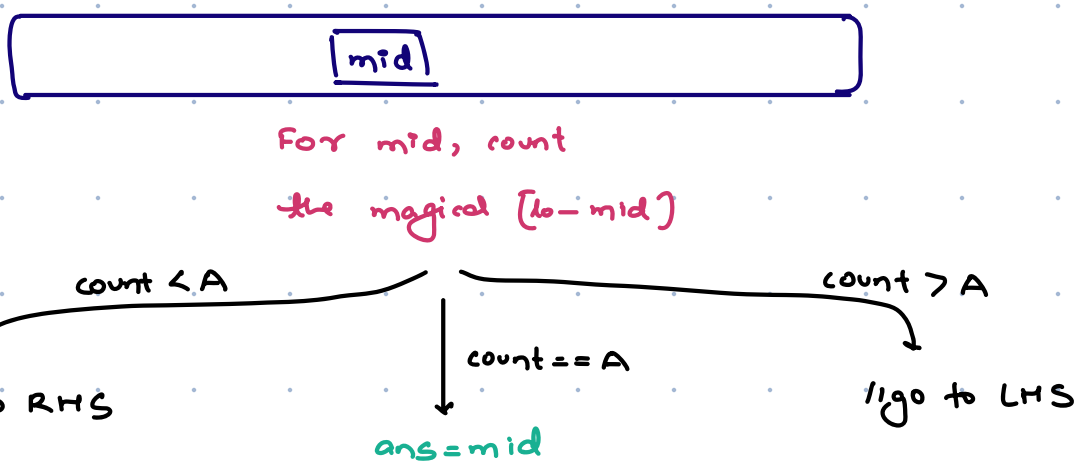
* B = 9
  C = 12
  } find the count of magical no. from 1 to 100

$$\Rightarrow \frac{100}{9} + \frac{100}{12} - \frac{100}{LCM(9,12)}$$

$$= 11 + 8 - 2 \rightarrow \underline{\underline{17}}$$

$$LCM(a,b) = \frac{a * b}{GCD(a,b)}$$

* Back to orginal question



B = 5 C = 7 A = 3

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

lo	hi	mid	count magical no. ≤ A
1	15	8	$\frac{8}{5}+\frac{8}{7}-\frac{8}{35}=1+1=2<3$
			// lo = mid + 1
9	15	12	$\frac{12}{5}+\frac{12}{7}-\frac{12}{35}=2+1=3==3$
			ans = 12
			hi = m − 1
9	11	10	$\frac{10}{5}+\frac{10}{7}-\frac{10}{35}=2+1=3==3$
			ans = 10
			hi = m − 1
9	9	9	$\frac{9}{5}+\frac{9}{7}-\frac{9}{35}=1+1=2<3$
			lo = m + 1
10	9	→	Stop

Ans = 10

```
lo = 1
hi = A * min(B,C)
int LCM = B*C / GCD(B,C)
while (lo <= hi){
    mid = (lo+hi)/2
    int count = countmagical (mid, B, C, LCM) = Mid/B + Mid/C - Mid/LCM
    if (count < A)  lo = m+1
    else if (count > A)  hi = m-1
    else {
        ans = m;
        hi = m-1;
    }
}
```

TC : O(log(min(B,C)) + log(A * min(B,C)))

SC : O(1)

* Median of an array → Middle of sorted Array

A[] = { 18 4 5 2 1 }

sort = { 1 2 4 5 18 } Ans = 4

Median of Two sorted arrays

A[] = {1, 4, 5}

B[] = {2, 3}

Ans[] = {1 2 3 4 5} Ans = 3

A[] = {4}

B[] = {1, 2, 3}

Ans[] = {1, 2, 3, 4} Ans = $\frac{2+3}{2}$ = 2.5

Q Given two sorted arrays, figure out the median of merged array.

Idea → Merge these two sorted arrays & return the middle of merged array.

Tc : O(n+m)

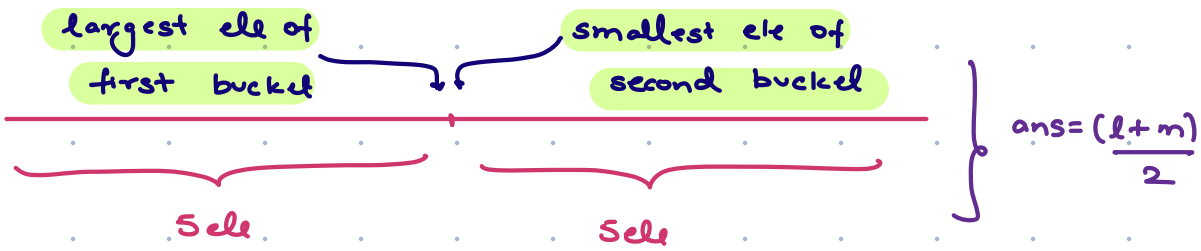
Sc : O(n+m)

A[] = {1 3 4 7 10 12}

B[] = {2 3 6 15}

size = 10

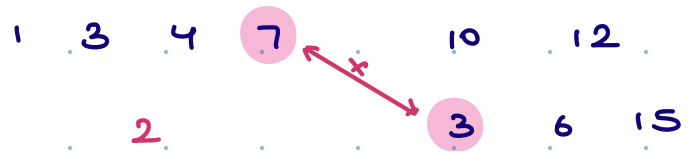
merged[] = {1 2 3 3 4 6 7 10 12 15}



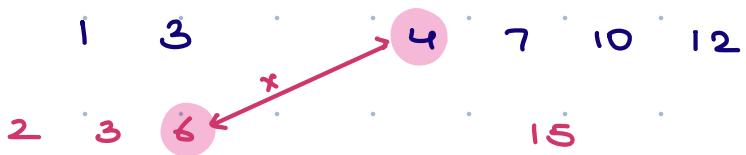
A[] = { 1 3 4 7 10 12 }

B[] = { 2 3 6 15 }

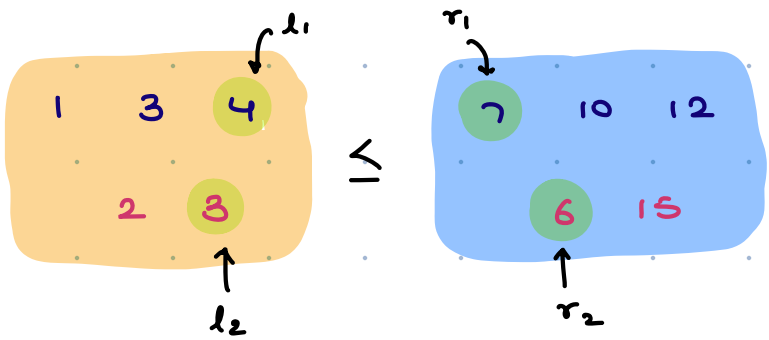
01. Scenario = 4 ele of 1st array



02 Scenario 2 = 2 ele from 1st array



03. Scenario 3 = 3 ele in 1st array



$l_1 \le r_2$

$l_2 \le r_1$

} correct bucketisation

Idea → Apply Binary search on smaller array & figure out how many ele we have to pick from the array in left bucket

l_1 r_1

A[ ] = { 7 12 | 14 15 }

0 1 2 3

l_2 r_2

B[ ] = { 1 2 3 | 4 9 11 }

Total ele = 10

No. of ele in one bucket = 5

search space = [0 4]

lo	hi	mid/cut	
0	4	2	→ 2 ele from A for LHS & 3 ele from B

$l_1 \leq r_2$ X : go to LHS

hi = mid - 1

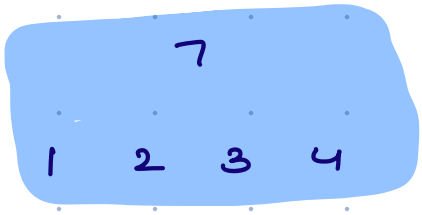
0	1	0	→ 0 ele from A for LHS & 5 ele from B

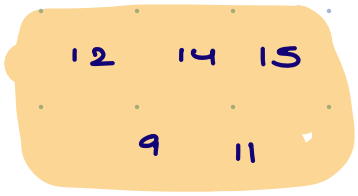
$l_2 \leq r_1$ X : go to RHS

lo = mid + 1

1	1	1	→ 1 ele from A for LHS & 4 ele from B

valid distribution





$$ans = \frac{max(l_1, l_2) + min(r_1, r_2)}{2}$$

lo = 0

hi = len of A // length of smaller array

while (lo ≤ hi){

int cut1 = (lo+hi)/2 // no. of ele from A for LHS

int cut2 = $\frac{(n+m+1)}{2}$ − cut1 // no. of ele from B for LHS

l_1 = (cut1 == 0) ? −∞ : A[cut1 − 1];

l_2 = (cut2 == 0) ? −∞ : B[cut2 − 1];

r_1 = (cut1 == n) ? ∞ : A[cut1]

r_2 = (cut2 == m) ? ∞ : B[cut2]

TC : O(log(search space))

SC : O(1)

if ($l_1 \le r_2$ && $l_2 \le r_1$){

if (n+m % 2 == 0){

return $\left(\frac{max(l_1, l_2) + min(r_1, r_2)}{2}\right)$

}

```
    else return max(l1, l2);
}
else if (l1 > r2) { hi = cut1 - 1; }   // go for less ele of A
else lo = cut1 + 1                     // go for less ele of B
                                       // which means more
                                       // elements of A
}
```