Today's Content:

1) → A^{Th} magical number

2) → k^{Th} index element in unsorted array.

- ↳ google/amazon/GfG/
- ↳ 2/3/4

3) → Simple 2 pointer

A^{Th} magical number

Given A, B, C, find A^{th} magical number

Note: a number is said to be magical number, if it is divisible by B or C

Eg1: B = 2, C = 3, A = 8^{Th} magical number

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

cnt = 1 2 3 4 5 6 7 8

ans = 12

8^{Th} magical

Idea:

1) Iterate from [1 - A] * wrong

2) Iterate till count becomes = A^{Th}

[1 - keep iterating]

magical number is a multiple of B or C

until get A^{Th} magical number

Eg1: A^2B, B = 2, C = 3

[1 — 16] 24]

8^{th} 2

Eg2: A = 10 | B = 4, C = 3 |

[1 → 30]

3^{rd} 10

Eg3: A = 5, B = 5, C = 8

[1 - 25]

5^{Th} magical number

// generalize:

A, B, C

[1 — A^{Th} * min(B, C)]

{Multiple of B or C}

Q) Given B, C, n, find nubers of magical nubers [1 n]

1) B = 3, C = 5, n = 35 {

[1 - 35] → 35/3 + 35/5 − 35/15 → {11 + 7 − 2} → 16

mul 3 mul 5

Comm: mul of 3 & 5

2) B = 9, C = 12 n = 100

[1 - 100] → 100/9 + 100/12 − 100/36 → {11 + 8 − 2} → 17

mul 9 mul 12

Comm mul of (9 & 12)

LCM(9, 12) → 36

magical [1 - x]

count(B, C, x)

return x/B + x/C − x/LCM(B,C)

}

LCM(B, C) {

Note:

B * C = LCM(B,C) * GCD(B,C)

}

Ex: A = 10, B = 2, C = 8

Search space [1, 20]

If 10th magical number ?

16 → [1 16] → $16/2 + 16/3 - 16/6 = 11 > 10$

How many magical ?

$8 + 5 - 2$

17 18 19 ... x x x

magic > 10 > 10 > 10 → [go to left]

10th magical

8 → [1 - 8] → $8/2 + 8/3 - 8/6 = 5 < 10$

— . x x x

— . 5 6 7 → [go to right]

< 10 < 10 < 10 magic

Pseudo code

TC: $[lo, hi] \Rightarrow [hi - lo + 1] \Rightarrow$

Given B, B, C

lo = 1, hi = A * min(B, C)

ans =

while (lo <= hi) {

mid = (lo + hi)/2

No: of magical [1 - mid]

n = count (B, C, mid) ⇒ magical numbers ↳ [1 mid]

if (n == A){

ans = mid; → Issue

hi = mid - 1 → useful

}

else if (n > A){

// goto left

hi = mid - 1

}

else{

lo = mid + 1

}

}

return ans

Trace:

B = 5, C = 7, A = 3

[1 15] mid = 8 [1 mid]

l = 1, h = 15, 2 < A, goto right

9 15 12 3 == A

return 12

12 is not even magical number

left

l	h	m	
1	15	8	
9	15	12	ans = 12
9	11	10	ans = 10
9	9	9	goto right
10	9		Break

#magical

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
#count: 0	0	0	0	1	1	2	2	2	3	3	3	3	4	5

// precompute gcd & LCM values

```
lo = 1,   hi = A * min(B, C)
ans =
while ( lo <= hi) {
    mid = (lo + hi)/2
    // No: of magical [1 - mid]
    x = count(B, C, mid)
                ↳ [1  mid]
    if ( x == A) {
        ans = mid;      } → Issue
        hi = mid - 1    } → useful
    }
    else if ( x > A) {
        // goto left
        hi = mid - 1
    }
    else {
        lo = mid + 1
    }
}
return ans
```

388) Given N array distinct unsorted array elements

find k^{th} index pos in it's sorted form /

Note: We cannot modify array / You cannot use Extra Space

Ex1: ar[5] = {2, 8, 3, 11, 14}

k = 2

0 1 2 → ans = 8

{2, 3, 8, 11, 14}

Ex2: ar[9] = {11, 24, 18, 3, 5, 27, 34, 9, 40}

0 1 2 3 4 5 6 7 8

k^{th} → {4 index} less: 3 5 4

ans = 18

0 1 2 3 → 4 Elements

a_0 a_1 a_2 a_3 < a_4

Atleast:

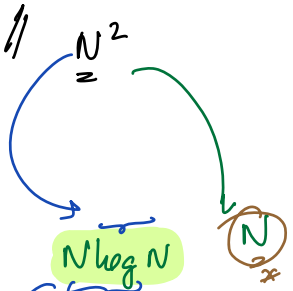
1) Merge k times & get mins
   - We cannot modify
   - No Extra Space
2) Quick Sort / Selection Sort / Min Heap / *

Pseudocode

1) Iterate on array
   - for every ar[i], get number of Elements are less than ar[i], (#count < ar[i]) == k

TC: O(N²) SC: O(1)

// N^2



Idea: Sort

Binary Search ?

1) Target → ✓

2) Search Space → ans should present
→ Divide & discard

→ [whole Array] } ans ✓
[0 or 1, N] } mid = N/2
* no discard

→ [0, k] *

→ [min max]

3) [min - max]

	0	1	2	3	4	5	6	7	8
k = 4	ar[] = {	11,	24,	18,	3,	5,	27,	34,	9,	40 }

l	h	mid	(#count Elem < mid)
3	40	21	< 21 : 5 ≠ 4, 5 > 4 ?

21 22 23 24 . . . goto left

* x x x x x x

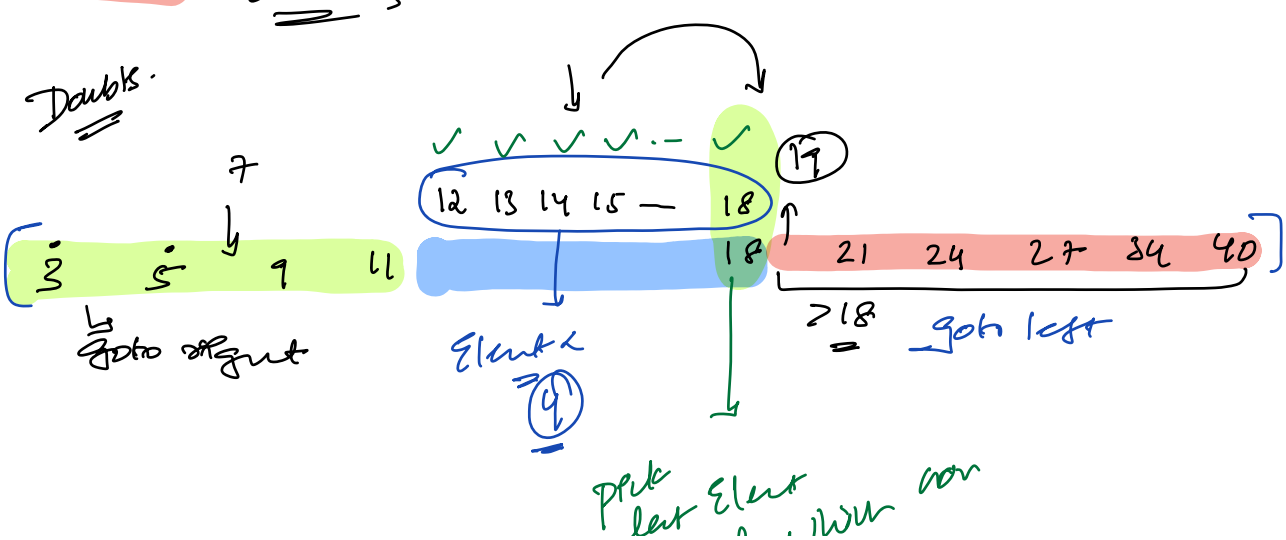
3	20	11	< 11 : 3 ≠ 4, 3 < 4 ?

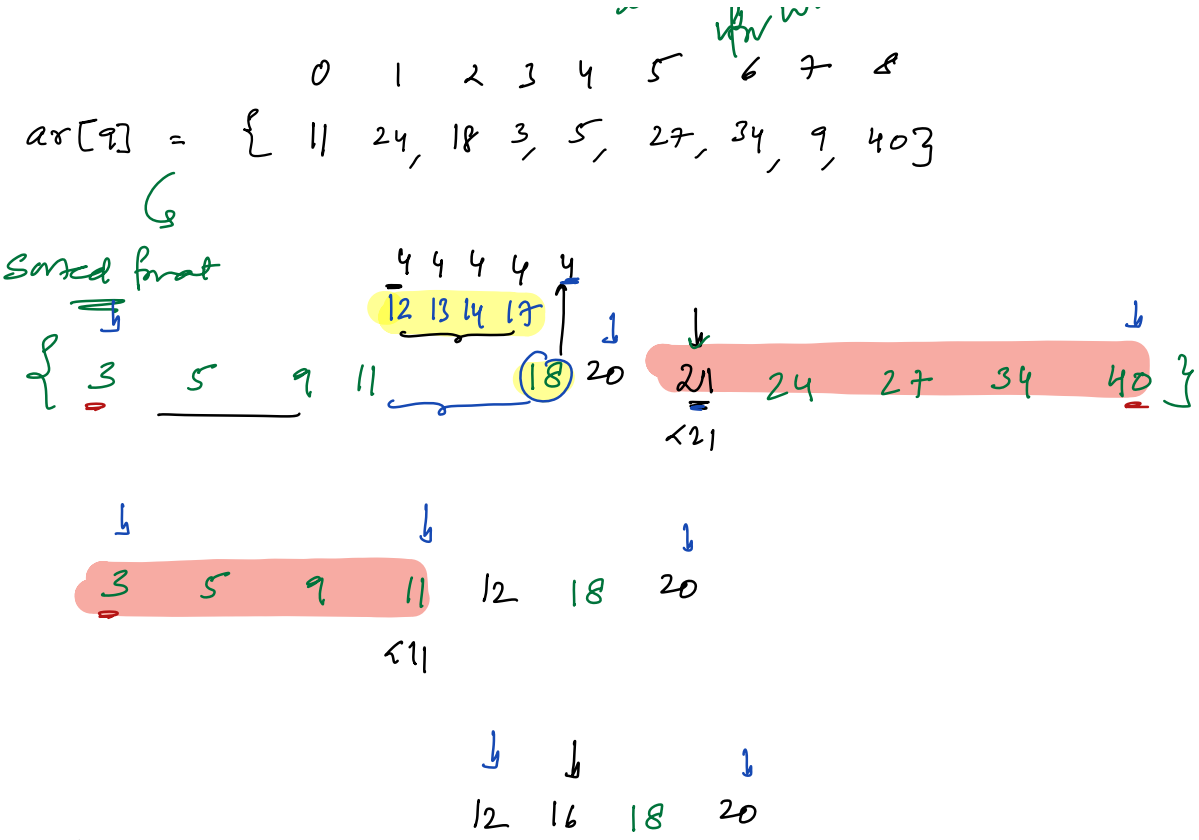
7 8 9 10 11 goto right

✓ x x x *

12	20	16	< 16 : 4 == 4 ans = mid lo = mid+1
17	20	18	< 18 : 4 == 4 ans = 18 lo = mid+1
19	20	19	< 19 : 5 > 4 hi = mid-1
19	18	= {Break}	

Double.





// Pseudocode :

lo = min(ar[]), hi = max(ar[]), ▷ given k

while(lo <= hi){

mid = (lo + hi)/2

x = {countless(Ar[], mid)} ▷ count no: of Elents le that mido

if(x == k){ ▷ Doubt

ans = mid; lo = mid+1

}

else if(x < k){

lo = mid+1 }

else{

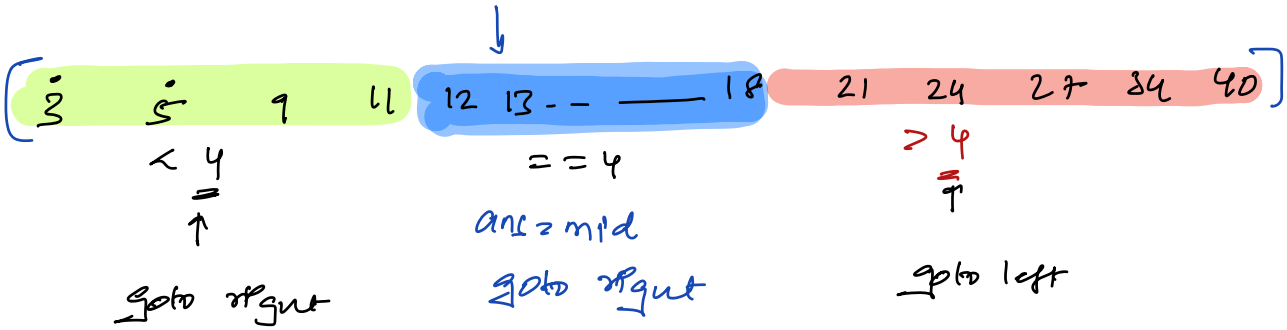
hi = mid-1 }

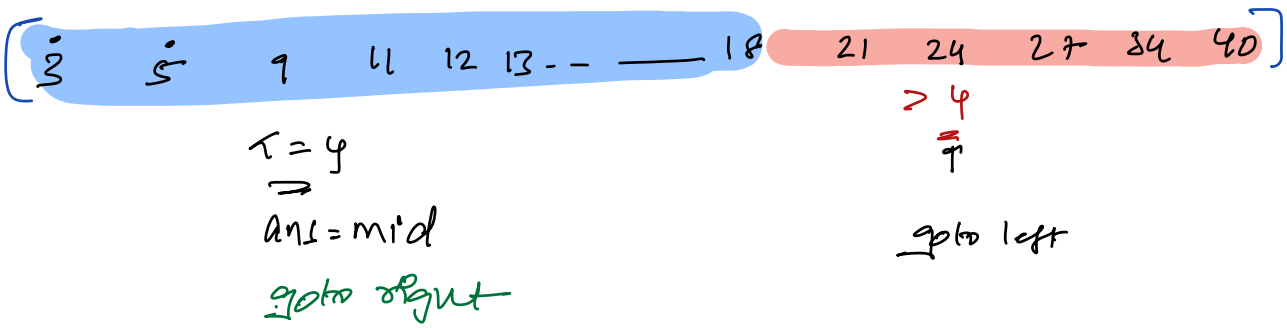
}

return ans

TC: $N * \log_2^{(Max - Min + 1)}$

SC: O(1)


3 5 9 11 12 13 -- ---- 18 21 24 27 34 40
< y
goto right
== y
ans = mid
goto right
> y
goto left


3 5 9 11 12 13 -- ---- 18 21 24 27 34 40
> y
goto left
= y
ans = mid
goto right

Q) ar[10]: { 20, 13, 13, 6, 3, 6, 13, 24, 13, 40 }

k = 5 → (TODO debug) → (Elements not distinct)

↳ (Pseudocode if ar[] elements are not distinct & if we need to find k^{th} order element in merged form)

lo = min(ar[]), hi = max(ar[]),

ans = lo

while (lo <= hi) {

  mid = (lo + hi)/2

  x = {countLess(ar[], mid)

  if (x <= k)

    ans = mid; l = mid+1

  else

    h = mid-1

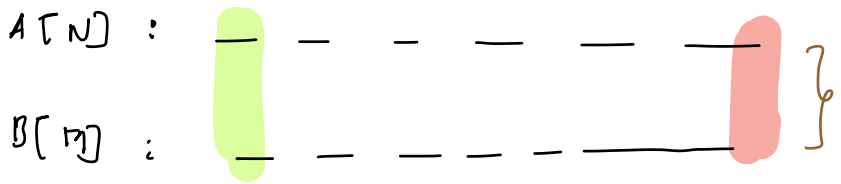
}

return ans

works if data is repeating

one of many useful

48) // Given 2 sorted Arrays find k^{th} pos in merged form.



find k^{th} pos in merged form

```
lo = min(A[0], B[0])      hi = (A[N-1], B[N-1])

while ( lo <= hi ) {

    int mid = (lo+hi)/2

    { x = countSmallerLen(A[], mid, N)
      x = x + countSmallerLen(B[], mid, M)

    if ( x <= k )
        ans = mid; l = mid+1
    else
        h = mid-1
}

return ans
```

same condition as before

TC : $\log_2(hi-lo+1) * \log_2(N)$

TODO: countSmallerLen(A, mid) : get a sorted array A[]

get no: of Elements less < mid : (logN)

SQ) Given N sorted arrays, Each of size M, find k^{th} index pos in it's sorted form.

lo = min( ) hi = max( )

```
while (lo <= hi){
    int mid = (lo+hi)/2
    c = 0
    for(i=0; i<N; i++){
        c = c + countSmallers(mat[i], mid, M)
    }
    if (c <= k){
        ans = mid; l = mid+1
    }
    else{
        h = mid-1
    }
}
return ans;
```

i^{th} row

same condition as above

TC: $\log_2(hi-lo+1) * [N * \log M]$

SC: O(1)