Todays Content:

- Adhoc Problems
- 22nd → 26th vacation

20th Thursday : _____

21 : Friday

22 : Satu

23 : Sun

24 : Mond

25 : Tue

26 : Wed

27 : Thursday : Session

18) Give unsorted arr[] of **N distinct elements**,
find k^{th} index pos in it's sorted form {amaj m}

Note: We cannot modify arr[] / We cannot use extra space

Ex1: arr[5] = { 2 8 3 11 14 } (idx: 0 1 2 3 4) **k = 2 ans = 8**
(2 → idx 0, 8 → idx 2, 3 → idx 1)

Ex2: arr[9] = { 11 24 18 3 5 27 34 9 40 } (idx: 0 1 2 3 4 5 6 7 8) **k = 4 ans = 18**
(11 → idx 3, 18 → idx 4, 3 → idx 0, 5 → idx 1, 9 → idx 2)

Idea: find k^{th} index element in sorted arr[]

0 1 2 ... k-1 k

$[a_0\ a_1\ a_2\ ..\ a_{k-1}] < a_k$

Obs: no: of elements less than that == k

arr[9] = { 11 24 18 3 5 27 34 9 40 } k = 6 (idx: 0 1 2 3 4 5 6 7 8)

ele < arr[i] =	3	5	4	0	1	6
	*	*	*	*	*	✓

Idea: For every arr[i], iterate & calculate no: of elem < arr[i] == k

Pseudo Code:

int k^{th}smallest(int arr[N], int k){ TC: O(N²) SC: O(1)

```
    for(i=0; i<N; i++){
        if(cless(arr[], arr[i]) == k){
            return arr[i]
        }
    }
}
```

→ Implement cless on your own

Idea 2: Binary Search

a) Target : k^{th} smallest element

b) search space : arr[]

c) We are not able to discard arr[] *

→ Change Search Space : [min of arr[] , max of arr[]]

Eg :

	0	1	2	3	4	5	
arr[5] = {	4	1	5	15	6	2	} k = 3

l	h	m	# count of arr[] ≤ m				
1	15	8	5 > 3	8	9	10	.... goto left
				F	F	F	h = m-1
1	7	4	2 < 3	.. 2	3	4	goto right
				F	F	F	l = m+1
5	7	6	4 > 3	goto left..			h = m-1
5	5	5	3 == 3	return 5			

Idea 2 : Pseudo Code :

```
int  k^th smallest ( int ar[N], int k){   TC: [log2 (max - min + 1)] * N
    l = min of ar[], h = max of ar[]  ans = ____      SC: O(1)
    while ( l <= h) {
        m = (l + h)/2
        // m is k^th index elem in sorted ar[]?
        // calalate no: of elements < m
        int  c = countless ( ar[], m)
        if ( c == k) {
            ans = m;
            l = m+1
        }
        if ( c < k) { // goto right
            l = m+1
        }
        else { // goto left
            h = m-1
        }
    }
    return ans;
}
```

Note: If data repeats above logic fails.

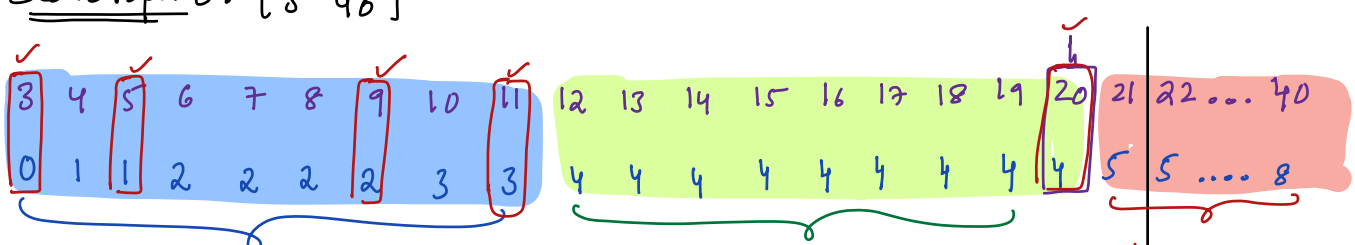
Edge Case:

Ens: { 11 24 20 3 5 27 34 9 40 } k=4

l	h	m	# count of arr[] < m	
3	40	21	5 > 4	21 22 23 .... 40 goto left F F F F h = m-1
3	20	11	3 < 4	... 9 10 11 goto right l = m+1 F F F
12	20	16	4 == 4	16 is ans, but it's not even in arr[]

{ 11 24 20 3 5 27 34 9 40 }

SearchSpace: { 3 40 }



class < k
goto right

class = k
ans = m;
goto right;

class > k:
goto left

Q) Elements can repeat :

ar[8] = { 15 4 15 10 16 19 10 15 } k = 4

l	h	m	# count of ar[] < m
4	19	11	3 < 4 : ans = 11, goto right l = m+1
12	19	15	3 < 4 : ans = 15, goto right l = m+1
16	19	17	7 > 4 : goto left
16	16	16	5 > 4 : goto left
16	15	} Break	

SearchSpace [4 19] K = 4

4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	1	1	1	1	1	3	3	3	3	3	6	7	7	7

ans = mid
goto right

c > k :
goto left

Repeated elements:

```
int  k^th smallest (int ar[N], int k){   TC: [log2 (max-min+1)] * N
    l = min of ar[], h = max of ar[]  ans = ____   SC: O(1)
    while (l <= h){
        m = (l+h)/2
        // m is k^th index elem in sorted ar[]?
        // calalate no: of elements < m
        int c = countless (ar[], m)
        if (c > k){
            //goto left
            h = m-1
        }
        else{
            ans = mid
            l = mid+1
        }
    }
    return ans;
}
```

Note: Above logic works for all cases

8:35 —lomi→ 8:45

208) Given 2 sorted arrays[], find k^{th} pos in overall sorted data

N

	0	1	2	3	4	5	6	7
A[8] = {	3	3	6	7	7	11	14	17 }

M

	0	1	2	3	4	5	6
B[7] = {	2	2	10	10	13	20	20 }

k = 8

Idea1: Merge 2 sorted arr[] TC: O(N+M)

Idea2: Using Binary Search:

Target : k^{th} pos element

Search Space: min of (A[], B[]) max of (A[], B[])

Discard : Yes

l	h	m	# count of <m in (A, B)	k=8
2	20	11	9 > 8	h = m-1
2	10	6	4 <=8	ans=6, l=m+1
7	10	8	7<=8	ans=8 l=m+1
9	10	9	7<=8	ans=9 l=m+1
10	10	10	7<=8	ans=10 l=m+1
11	10	{ break }		

```
int countless sorted (int ar[N], int k){
    // given sorted arr[], calculate no: of elements < k
       using binary search  TC: O(log N)  SC: O(1)
}

int kth smallest (int ar[N], int br[M], k)

    l = min(ar[0], br[0])   h = max(ar[N-1], br[M-1])   ans = ____

    while (l <= h){

        m = (l + h)/2
        // m is kth index elem in sorted arr[]?
        // calculate no: of elements < m
        int c = countless sorted (ar[], m)   } log N
        c = c + countless sorted (br[], m)   } log M

        if (c > k){ //goto left
            h = m-1
        }
        else{
            ans = mid
            l = mid+1
        }
    }
    return ans;
}
```

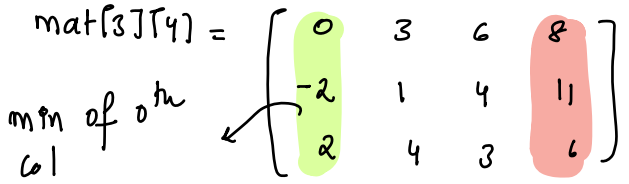
TC: $\log_2^{(hi-lo+1)} * \left[\log_2^N + \log_2^M\right]$

↳ TC: $\log_2^{[N+M]}$ TODO

↳ median of 2 sorted arrays

// Given mat[N][M] every row sorted, find k^{th} pos in overall sorted data

mat[3][4] =



int k^{th} smallest (int mat[N][M], int k) {

```
l = min of 0th col   h = max of M-1th col

while (l <= h) {                          TC: log2(h-l+1) * [N * log M]   SC: O(1)
    m = (l+h)/2
    // m is kth indexed elem in sorted arr?
    // calculate no: of elements < m
    c = 0
    for(i=0; i<N; i++) {
        c = c + countLessSorted(mat[i], m)
    }                                  // passing ith row as parameter
                                       // each row contains M elements
    if (c > k) { //goto left
        h = m-1
    }
    else {
        ans = mid
        l = mid+1
    }
}
return ans;
}
```

// Give mat[N][M], every row is sorted, find median? Interview

// Say N*M = all elem = 0 1 2 ... 10 . 11 .. 20

// ele at index 10 is median, k=10

// Say N*M = 10 ele = 0 1 2 3 4 5 6 7 8 9

// avg of ele at index 4 5, k=4, k=5

```
if ( (N*M) % 2 == 1) {
    // only 1 median
    kth smallest ( mat[N][M], (N*M)/2 )
}
else
    // 2 median find = (N*M)/2
    [ kth smallest ( mat[N][M], find)
                    +
      kth smallest ( mat[N][M], find-1) ] / 2
}
```

⇒ $\left[l + \frac{(h-l)}{2} \right]$ to avoid overflows

⇒ $\left[\frac{l+h}{2} \right]$ at times it can overflow