Todays Content:

- Adhoc Problems
- 22nd → 27th vacations

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

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

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

Ex2: arr[9] = { 11, 24, 18, 3, 5, 27, 34, 9, 40 } k = 4, ans = 18
(11 → 3rd, 18 → 4th, 3 → 0th, 5 → 1st, 9 → 2nd)

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

index: 0 1 2 3 ... k-1 | k
a_0 a_1 a_2 a_3 ... a_{k-1} < a_k

Obs: #elements less than that in arr[] == k, ⇒ ele is k^{th} index element

arr[9] = { 11, 24, 18, 3, 5, 27, 34, 9, 40 } k = 6
return 27

	0	1	2	3	4	5	6	7	8
arr[]	11	24	18	3	5	27	34	9	40
#counts of elements < arr[i]	3	5	4	0	1	6 == 6			

Idea: For every element, calculate no: of elements less than itself == k

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

```
for(i=0; i<N; i++){
    // count no: of elements < arr[i]
    if( cless(arr[], arr[i]) == k){
        return arr[i]
    }
}
```

TODO: TC: O(N)

```
int cless(int arr[], int ele){
    // no: of element < ele
    in arr[]
}
```

Idea 2: Binary Search

a) Target : k^{th} finden elem

b) SearchSpace : low = 0, high = N-1 $\Leftrightarrow$ low = min arr[], high = max arr[]

c) discard : With above search space, we cannot discard we cannot apply binary search,

Note: Change search space?

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

l	h	m	# count of arr[] < m
1	15	8	# count < 8 : 5 > 3 goto left h = m-1
1	7	4	# count < 4 : 2 < 3 goto right l = m+1
5	7	6	# count < 6 : 4 > 3 goto left, h = m-1
5	5	5	# count < 5 : 3 == 3 return 5

8 9 10 11 .... : F F F F

... 2 3 4 : F F F F

6 7 ... : F F F

Note: If count of ele < mid == k, it's not a guarantee that mid is final ans, hence we cannot directly return mid

Complete Idea:

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

l	h	m	# count of arr[] < m	
3	40	21	# count<21 = 5 > 4 goto left h = m-1	21 22 23 ... F F F
3	20	11	# count<11 = 3 < 4 goto right l = m+1	... 9 10 11 F F F
12	20	16	# count<16 = 4 = 4	ans = 16, goto right
17	20	18	# count<18 = 4 = 4	ans = 18, goto right
19	20	19	# count<19 = 4 = 4	ans = 19, goto right
20	20	20	# count<20 = 4 = 4	ans = 20, goto right
21	20	break	return ans = 20	

0	1	2	3	4	5	6	7	8	
{ 11	24	20	3	5	27	34	9	40 }	k=4

SearchSpace: lo = 3 h = 40, { #no: of elements less than that }

3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22 ...	40
0	1	1	2	2	2	2	3	3	4	4	4	4	4	4	4	4	4	5	5	8

class < k → No: of elements less than that → goto right

class == k → ans = mid/update → goto right

class > k → goto left

Elem:

3	5	9	11	12	13	19	20	21	24	27	34	40
				4	4	4	4	5				

Idea2: Pseudo Code:

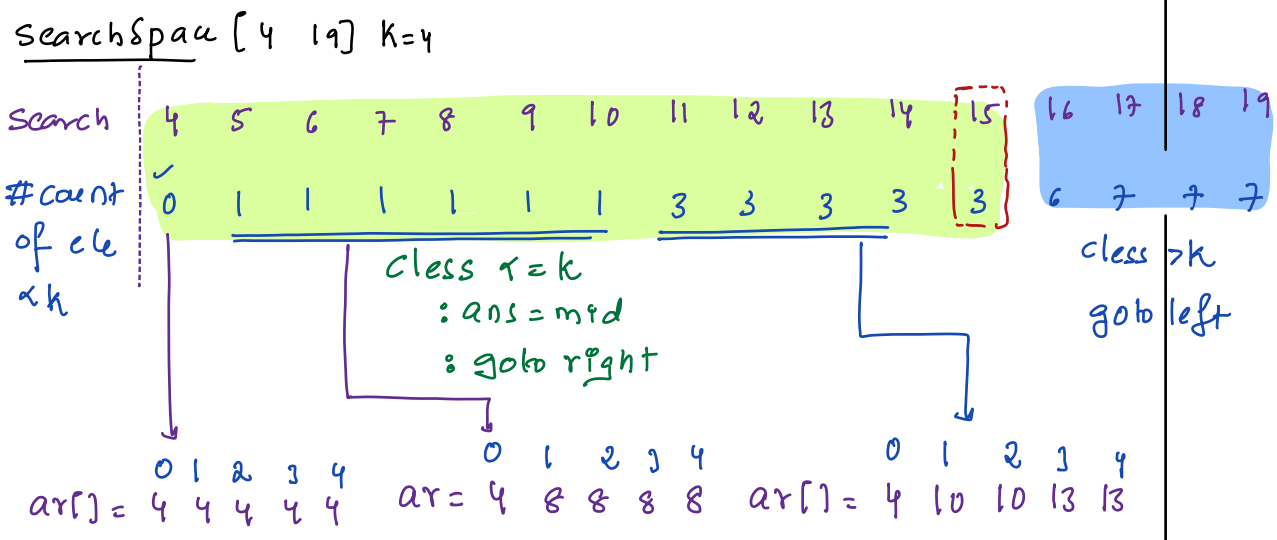
```
int kthIndex(int ar[N], int k){        TC: O(N * log2^(hi-lo+1)) SC: O(1)
    lo = min ar[]  hi = max ar[] => TC: O(N)
    while(lo <= hi){                   #TC: N * ⌈log2^(hi-lo+1) * N⌉
        m = (lo+hi)/2
        // Calculate no: of elements < m in ar[]
        c = cless(ar[], m) // count of elements < m  TC: O(N)
        if(c == k){
            ans = m; //goto right
            l = m+1
        }
        else if(c < k){ //goto right
            l = m+1
        }
        else{ // c > k, goto left
            h = m-1
        }
    }
    return ans;
}
```

Note: Above logic fails, if data repeats

Q) Elements can repeat :

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

l	h	m	# count of ar[] < m
4	19	11	#count<11 = 3 < 4 , ans = 11 goto right } l = m+1
12	19	15	#count<15 = 3 < 4 ans = 15, goto right } l = m+1
16	19	17	# count<17 7 > 4 , goto left h = m-1
16	16	16	# count<16 6 > 4 , goto left h = m-1
16	15	break return ans = 15	



Repeated elements:

```
int kthSmallest(int ar[N], int k){   TC: O(N * log2^(hi-lo+1)) SC: O(1)
    lo = min ar[]   hi = max ar[] => TC: O(N)
    while(lo <= hi){                          #TC: N * ⌈log2^(hi-lo+1) * N⌉
        m = (lo+hi)/2
        // Calculate no: of elements < m in ar[]
        c = cless(ar[], m) // count of elements < m   TC: O(N)
        if(c <= k){
            ans = m; // goto right
            l = m+1
        }
        else{ // c > k, goto left
            h = m-1
        }
    }
    return ans;
}
```

Note: Above code works, even if ar[] elements distinct

Verify with TODO

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

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

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

k=8, ans = 10

Idea1: Merge 2 sorted arr[] & get overall sorted data & extract k^{th} index element

TC: O(N+M)

Idea2: Binary Search:

a) Target : k^{th} index element

b) SearchSpace : lo = min(A[0], B[0]) ; hi = max(A[n-1], B[m-1])

c) discard : ½s

l	h	m	# count of elements in A & B k = 8
2	20	11	# count ≤ 11 in A, B = 9 > 8 : goto left h = m-1
2	10	6	# count ≤ 6 in A, B = 4 < 8 : ans = mid, 6 goto right l = m+1
7	10	8	# count ≤ 8 in A, B = 7 < 8 : ans = mid, 8 goto right l = m+1
9	10	9	# count ≤ 9 in A, B = 7 < 8 : ans = mid, 9 goto right l = m+1
10	10	10	# count ≤ 10 in A, B = 7 < 8 : ans = mid, 10 goto right l = m+1
11	10		break, return ans = 10

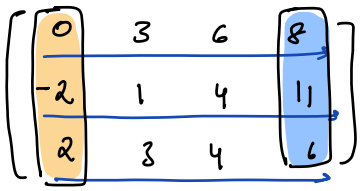
```
int clessSorted(int ar[N], int k) TC: O(logN) SC: O(1)
    // Calculate & return count of ele < k in given sorted arr
    using binary search : TODO
}

int kthEle(int ar[N], int br[M], k) TC: log2^(hi-lo+1) * [logN + logM]
                                    SC: O(1)
    lo = min(ar[0], br[0])  hi = max(ar[N-1], br[M-1])

    while(lo <= hi){
        mid = (l+h)/2
        # no: of less than mid in both array A & B
        // ele < M in sorted arr[] using binary search
        c = clessSorted(ar[], mid) TC: logN
        c = c + clessSorted(br[], mid) TC: logM
        if(c <= k){
            ans = mid ; // goto right
            l = mid + 1
        }
        else{ // c > k, goto left
            h = mid - 1
        }
    }
    return ans
}
```

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

mat[3][4] =



int k^{th} smallest (int mat[N][M], int k){ TC: $\log_2^{hi-lo+1} * [N * \log_2^M]$ SC: O(1)

lo = min of 0^{th} col in mat[][]

hi = max of $M-1^{th}$ col in mat[][]

while (lo <= hi){

mid = (l+h)/2

#get no: of elements <= mid in every row binary search

c = 0;

for(i = 0; i < N; i++){

c = c + classSorted(mat[i][], mid) TC: $\log_2^M$

//no: of elements <= mid in i^{th} row

// size of row = M

}

if(c <= k){

ans = mid ; //goto right

l = mid + 1

}

else{ // c > k, goto left

h = mid - 1

}

}

return ans

Topics to revise:

a) Hashmap
b) Strings
c) LinkedList
d) Stacks