Today's content.

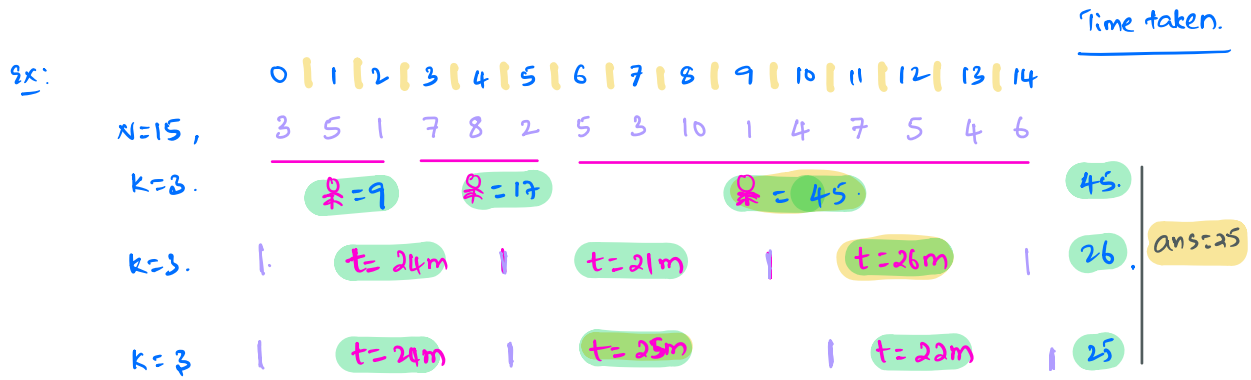
(i) Workers allocation

(ii) Aggressive cows.

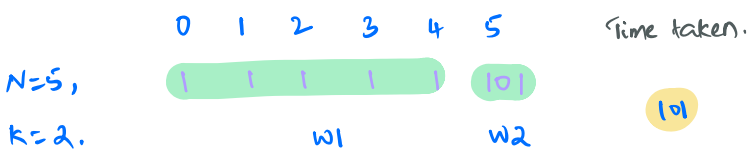
10. Given N tasks, k workers and time taken for each task, find min time in which we can complete all tasks.

Note: i) A single worker can only do continuous set of tasks
ii) All workers start their assigned task at same time
iii) A task can only be assigned to 1 worker.

Ex:



Ex:



Ideal: Brute force.

Q: In how many ways can I choose 2 sticks out of 14 sticks?

$^{14}C_2$.

Q: In how many ways can I choose (k-1) sticks out of (n-1) sticks?

$^{n-1}C_{k-1} * N$. ← TC.

↳ prefix sum ⇒. $N + {}^{n-1}C_{k-1} * k$ ← TC.

Idea 2: Binary search.

a) Target ⟶ min time.

b) search space ⟶ [low: max of all times , high: Sum of all times]

Ex1: ar : [10 2 9 8].

K=1, Time taken = 29.

Ex2: ar : [10 15 11 13]. (W1 W2 W3 W4)

K=4 : Time taken.

Discard.

Case 1: If we can finish the task in mid time.

ans = mid

goto left.

Case 2: If we cannot finish task in mid time.

goto right.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 | N=15, K=4.

3 5 1 7 8 2 | 5 3 10 1 4 7 5 4 6

[W_1 = 26min][W_2 = 30min.][W_3=15min] W_4=[ ] idle.

[W_1=9min][W_2][W_3][W_4]. ← Tasks left.

Can I finish all the tasks in 10 min?

. . . . 7 8 9 10

F F F F

Can I finish all the tasks in 30 min?

30, 31, 32, 33 - - - -

T T T T - -

Code:

```
def mintime(ar, k, n)        // n: Time for each task. k: # workers. n: #Tasks.
{
    l = max(ar), h = sum(ar), ans = sum(ar).
    while(l <= h)
    {
        m = (l+h)/2.
        // can we finish all the tasks in m time.
        if (check(n, k, ar, m))
            ans = m
            h = m-1   // goto left.
        else   // goto right
            l = m+1

    }
    return ans
}
```

TC: $N * \log_2(\#\text{elements})$

elements = h-l+1

= sum(ar) - max(ar).

```
def check(n, k, ar, t)
{
    c = 0, s = 0
    for i in range(0, N)
        s = s + ar[i]  // adding task time.
        if(s > t)
            c = c+1, s = ar[i]
            if(c == k) // utilized k workers, but tasks are left.
                return false
    return true
}
```

Idea for check fun.

N=15
K=4.
T=20.
C=0,

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	5	1	7	8	2	5	3	10	1	4	7	5	4	6
3	8	9	16	~~24~~ 8	10	15	18	~~28~~ 10	11	15	~~21~~ 7	12	16	22

C:C+1 C:C+1 C:C+1 C:C+1 left out.

Even after utilizing 4 people, tasks are left out, return false

---

N=15,
K=4.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	5	1	7	8	2	5	3	10	1	4	7	5	4	6

W1 W2 W3 W4. // 24 ✓
W1 W2 W3 W4 // 16 ✗.
W1 W2 W3 W4 // 20 ✗.
W1 W2 W3 W4 // 22 ✓
// 21 ✗.

l	h	m	Can I finish all tasks in m time	ans update.
10	71	40.	✓	ans = 40, goto left, h = m-1.
10	39	24	✓	ans = 24, goto left, h = m-1
10	23	16	✗	goto right, l = m+1.
17	23	20	✗	goto right, l = m+1
21	23	22	✓	ans = 22, goto left, h = m-1
21	21	21	✗	goto right, l = m+1
22	21	(l > h) ✗	break return 22.	

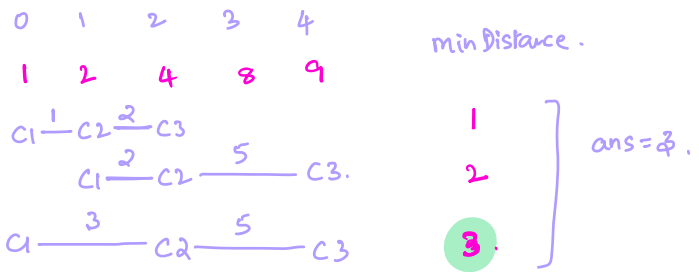
Break.
10:28:00
10:38:00
10:40:00

20. Given **k cows** and **N stalls**, all stalls are on **x-axis** at different locations, Place all cows in such a way that **minimum distance between the 2 cows is maximized**. (Keep the aggressive cows as far as possible)

Note: (i) Only **one cow** can be present at **one stall**.
(ii) **All cows** need to be **placed**, (N > K), All stalls positions are sorted.

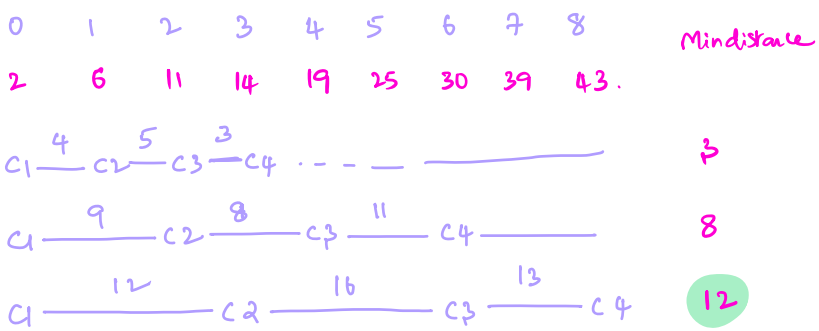
Ex1:

Stalls = 5
cows = 3



Ex2:

Stalls = 9
cows = 4



Idea 1: Given N stalls, ways to choose k stalls for k cows.

$^{N}C_{k} * k$.
↓ No. of ways
→ To calculate min distance for a selection.

Idea 2: Binary search.

a) Target : maximum min distance between 2 cows.

b) Search space :

low	high
min of adjacent elements difference.	stalls[N-1] - stalls[0].

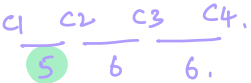
Ex1:

stalls[0] : [3 8 14 20].

k = 2 cows :

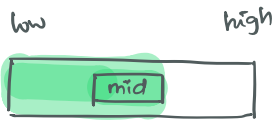
stalls[0] : [3 8 14 20].

k = 4 cows :



BS discard.

Case 1 : Say we can place all cows at a distance of "mid".



ans = mid

move right.

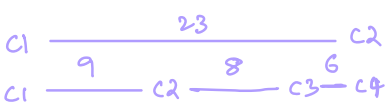
Case 2 : Say we cannot place all cows at a distance of "mid".

move left.

Stalls = 9

cows = 4

0	1	2	3	4	5	6	7	8
2	6	11	14	19	25	30	39	43.



C3, C4. X. → 20.

✓ for 5

Can I place cows atleast at 20 distance apart?

20	21	22	23
F	F	F	F

Can I place cows atleast at 5 distance apart?

2	3	4	5
T	T	T	T

Code:

```
def maxDistance(n, k, stalls)        # n -> no of stalls, k -> no of cows.
    l = min adjacent diff. in stalls[], h = stalls[n-1] - stalls[0], ans =
    while (l <= h)
        m = (l+h)/2
        // If you can place all cows at 'm' distance apart.
        if (check(n, k, stalls, m))
            ans = mid
            l = m+1
        else
            h = m-1
    return ans
```

(h-l+1).

TC: N * O(log$_2$(#elements))

m = 20.

```
def check(n, k, stalls, m)
{
    c = 1, j = 0
    for i in range(1, N)
        if (stalls[i] - stalls[j] >= m)
            c = c+1
            j = i
    return c >= k
}
```

0	1	2	3	4	5	6	7	8
2	6	11	14	19	25	30	39	43.
C1					C2			

Doubts

0	1	2	3	4	5	6	7	8
2	6	11	14	19	25	30	39	43.
C1			C2			C3		C4

l	h	m	CanI place all cows	update ans.
3	41	22	X	goto left, h = m-1.
3	21	12	✓	ans = 12, goto right, l = m+1
13	21			