Todays Content:

a) Workers allocation

b) Aggressive Cows

Binary Search?

a) Target b) Searchspace c) Discard or Not?

1. Need not be sorted

2. Need not be Input[]

Q1: Given N tasks, k workers & time taken for each task
find min time, in which we can complete all tasks

Notes: 1. A single work can only do continous set of tasks ✓
2. We cannot change order of tasks ✓
3. A task can only be assigned to 1 worker ✓
4. A worker can take multiple tasks ✓
5. All workers start their assigned tasks at same time

Eg:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
N=15 :	3	5	1	7	8	2	5	3	10	1	4	7	5	4	6	Time taken
k=3 :	$w_1 = 34$								$w_2 = 15$			$w_3 = 22$				34.
	$w_1 = 26$								$w_2 = 23$			$w_3 = 22$				26
	$w_1 = 24$								$w_2 = 25$			$w_3 = 22$				25: Final ans

Eg2:

	0	1	2	3	4	5	
ar[6] =	1	1	1	1	1	101	Time taken
k=2 :	$w_1 = 3$			$w_2 = 103$			103
	$w_1 = 5$					101	101 = ans

Idea: assign avg time:

In above example:

Total time = 106 k=2 avg time = 53

Issue: We cannot assign avg time

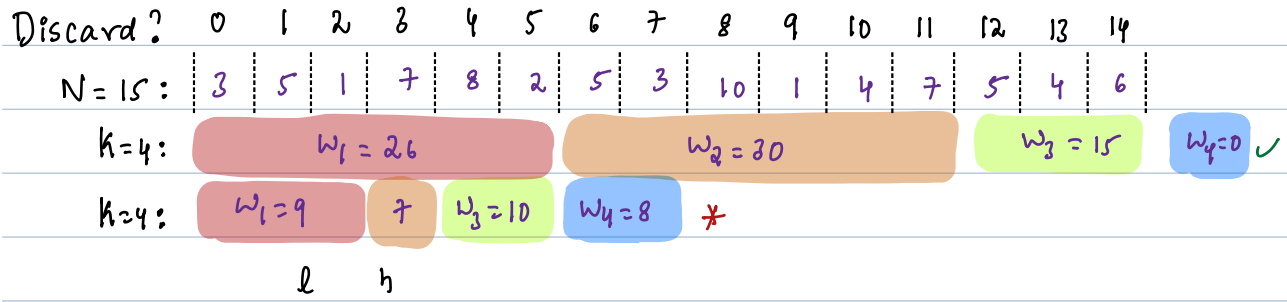
Let's Search:

Target: Min time to complete all tasks

SearchSpace: [l . . . . . h]

l: max time taken by task

h: sum of all time taken by tasks



SearchSpace: {10 71} // on total time taken to finish all tasks

Eg: Say mid lands at = 30. Can we finish all tasks by 30min?

30 31 32 33 . . . .
T T T T

Note: If we can do it 30, we can do more than 30 as well

ans = 30; goto left;

Eg: Say mid lands at = 12. Can we finish all tasks by 12 min?

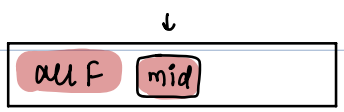
. . . 10 11 12
F F F

Note: If we cannot do it at 12 min we cannot do less than 12

goto right;

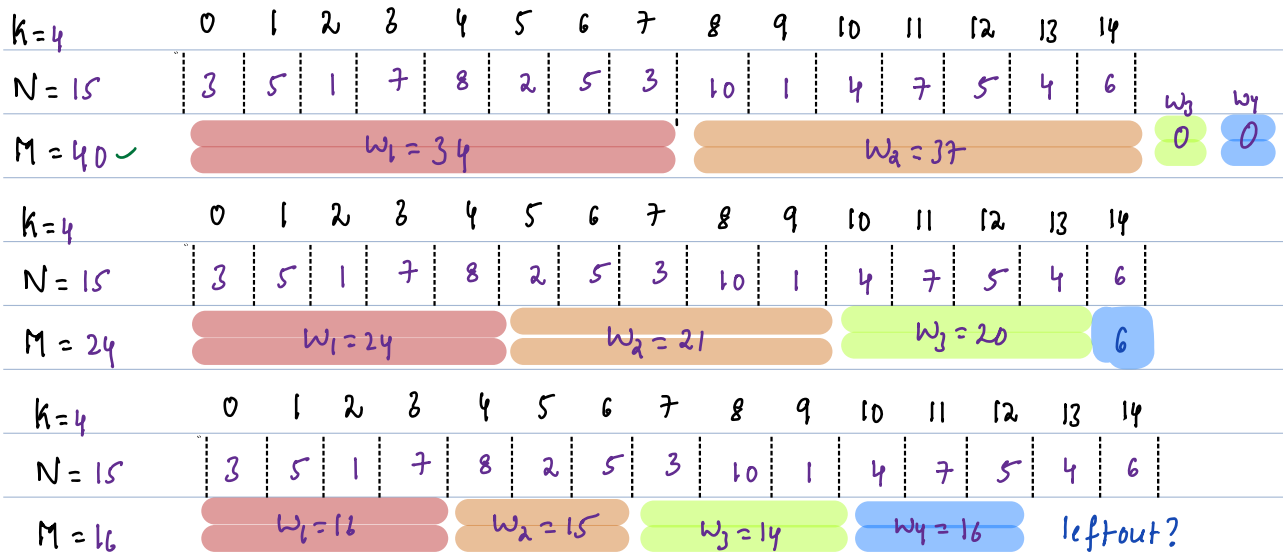
Discard:

if (finish task in mid time) { ans = mid, goto left}



if (cannot finish task in mid time) {goto right}

Check function Idea:



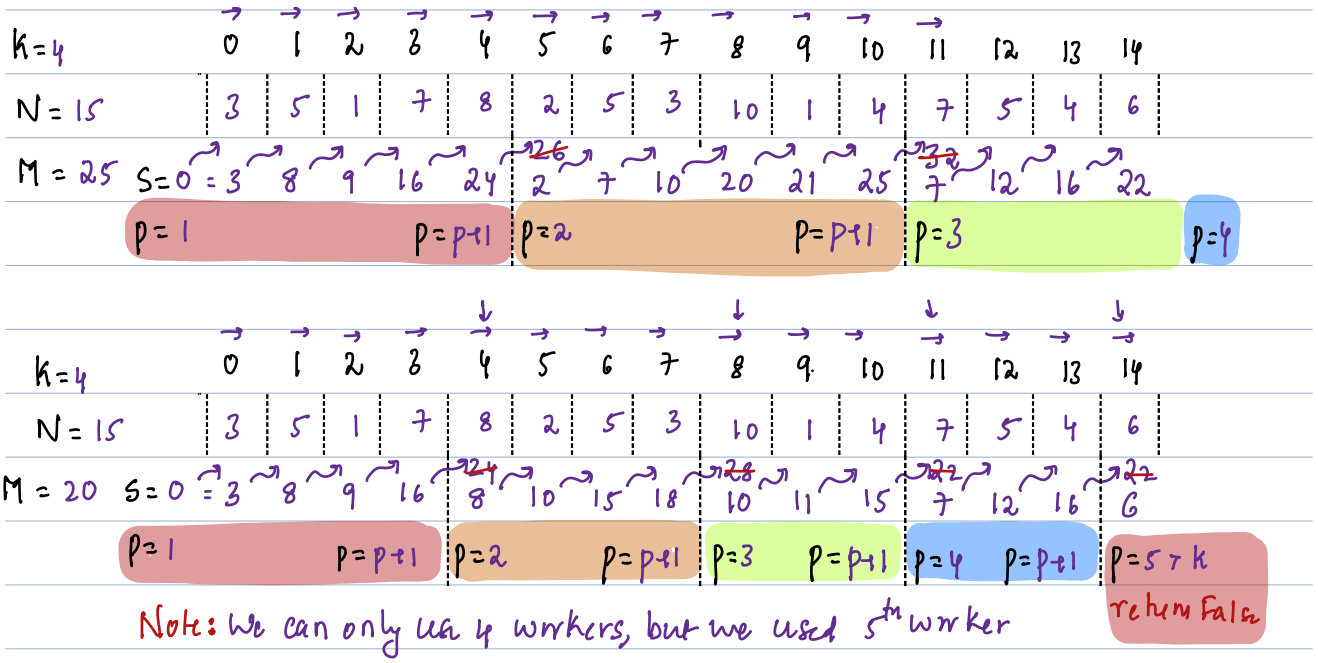
l	h	m: l+h/2	Can we finish tasks in m time	Update
10	71	81/2 = 40	Finish in 40 time	True ans = 40 h = m-1 goto left
10	39	49/2 = 24	Finish in 24 time	True ans = 24 h = m-1 goto left
10	23	33/2 = 16	Finish in 16 time	False l = m+1, goto right
17	23	soon.. {TODO}		

Pseudo Code Tasks workers

```
int minTime(int N, int k, int time[]){          Searchspace size = h-l+1
    int l = max of time[]   h = sum of all time[]   We apply BS on = h-l+1 ele
    int ans = h // Initialize with max              BS iterations = log2 h-l+1
    while(l<=h){
        int m = (l+h)/2
        //Check if we can finish all tasks in m time, using k workers?
        if(check(time[], k, m)){
            ans = m;  h = m-1; goto left
        }else{
            l = m+1; goto right
        }
    }
    return ans;
}
```

Final TC: $\log_2^{h-l+1} * N$ SC: O(1)

Check func:



Note: We can only use 4 workers, but we used 5th worker

```
boolean check(int time[], int k, int M){ TC: O(N) SC: O(1)
    int N = time.length;
    int S = 0, p = 1;
    for(int i=0; i< N; i++){
        S = S + time[i]
        if(S > M){ //allocation is incorrect
            p = p + 1;
        } S = time[i] // re-assigne ith task to new person
    }
    if(p > k){return false}
    else{return true}
}
```

2) Given k cows & N stalls, all stalls are on x-axis at different locations, Place all k cows in such a way that min distance between any 2 cows is maximized. Maximize min distance

Note1: In a stall only 1 cow can be present ✓

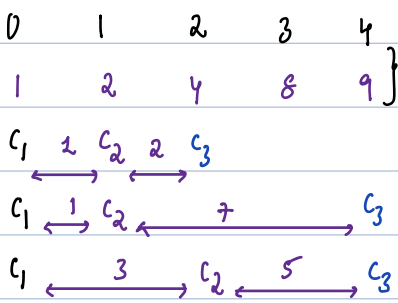
Note2: All cows have to placed, Stalls N > Cows k ✓

Note3: All stalls positions are sorted, if not sorted we can sort ✓

Ex1:

Stalls = 5 xanis[] = { 1 2 4 8 9 } mindist, Inc

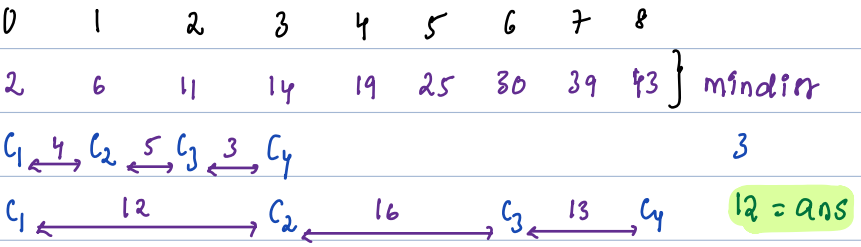
Cows = 3



Ex2:

Stalls = 9 xanis[] = { 2 6 11 14 19 25 30 39 43 } mindist

Cows = 4



Let's Search:

Target: Maximum distance by which we can separate any 2 cows

Searchspace: [l ..... h] // on distance between 2 cows

{ l: Min adj diff
  h: last - first } generalizations:

Discard?

		0	1	2	3	4	5	6	7	8
Stalls = 9	dist[] =	2	6	11	14	19	25	30	39	43

Cows = 4

M = 20 ✗ C_1 ←— 23 —→ C_2 C_3 C_4 : *

M = 7 ✓ C_1 ←9→ C_2 ←8→ C_3 ←11→ C_4

Searchspace: { l = 3, h = 41 }

En: Say mid lands at = 20. Can we place them atleast 20 dist apart?

20	21	22	23 ..
F	F	F	F

Note: If we cannot keep them at 20 dist apart, we cannot them at 21, 22 ...

goto left

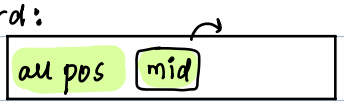
En: Say mid lands at = 7 Can we place them atleast 7 dist apart?

.... 4	5	6	7
T	T	T	T

Note: If we can keep them at 7 dist apart we can keep them at 6, 5, ...

ans = 7 goto right

Discard:



If (Can place cows atleast mid distance apart)

: ans = mid goto right

If (Cannot place cows atleast mid distance apart) : goto left;

Stalls Cows

```
int maxdist(int N, int k, int xanis[])
    Arrays.sort(xanis) // sort positions
    int l = minadjdiff in xanis[]  h = xanis[N-1] - xanis[0];
    int ans = l // initialize min
    while(l <= h){
        int m = (l+h)/2
        // check if we can separate cows atleast m dist apart?
        if(check(xanis[], k, m)){ // if possible
            ans = m;
            l = m+1 // goto right
        }else{ // Not possible m distance apart
            h = m-1; //goto left
        }
    }
    return ans;
}
```

Search space size = h-l+1

We apply BS on = h-l+1 elements

BS iterations = $\log_2^{h-l+1}$

Final TC: $\log_2^{h-l+1} * N$ SC: O(1)

Check function Idea: TC: O(N) SC: O(1) : Code TODO

	0	1	2	3	4	5	6	7	8
		→	→	→	→	→	→		
Stalls = 9 dist[] =	2	6	11	14	19	25	30	39	43
Cows = 4		c_1		c_2		c_3		c_4	
M = 8		p=1		p=2		p=3		p=4	: In loop :
		x=2		x=11		x=19		x=30	p == Cows : return true

	0	1	2	3	4	5	6	7	8
		→	→	→	→	→	→	→	→
Stalls = 9 dist[] =	2	6	11	14	19	25	30	39	43
Cows = 4		c_1			c_2			c_3	: After loop :
M = 15		p=1			p=2			p=3	p=3 < Cows : return false
		x=2			x=19			x=39	