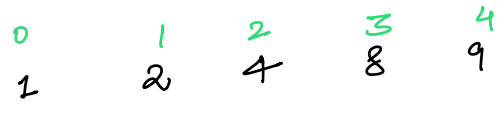
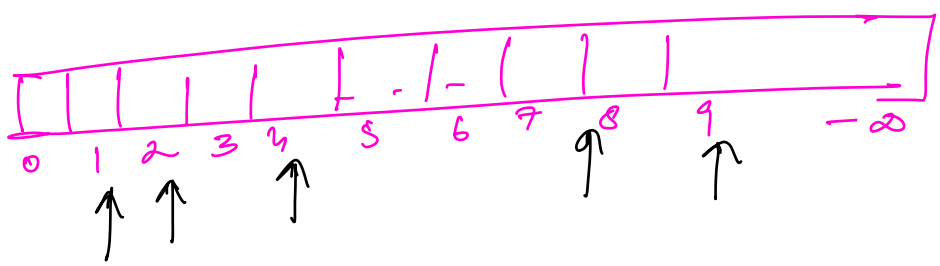
→ Aggressive Cows

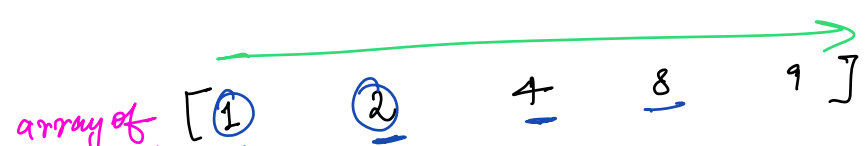
N cow stalls. K cows. (N ≥ K)

K=3.

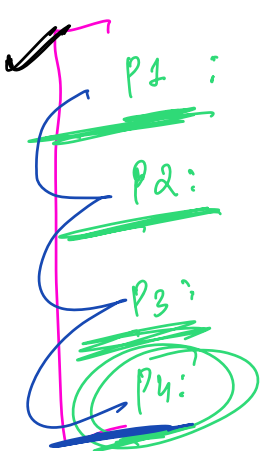


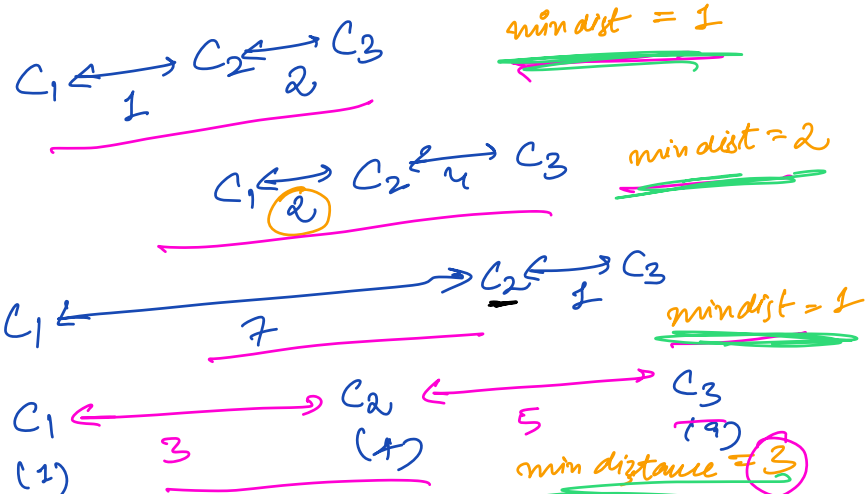


→ cows are very aggressive.



K=3.





Q. Given N cow stalls & M cows ($N \geq M$)

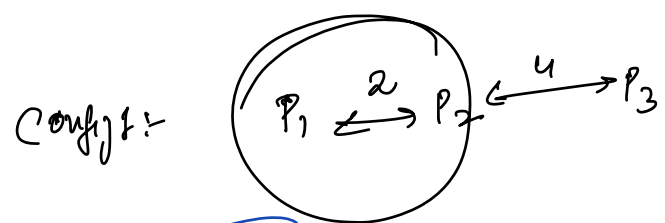
A[i] → position of each stall in ascending order

- You have to place all the M cows in these stalls.
- We have to find the max. distance b/w any two cows given that cows need to be placed as far as possible.

↓ → min dist b/w 2 adjacent cows.

~~Config1~~	: D_1	
~~Config2~~	: D_2	D_i
Config3	: D_3	
4	D_4	
5	D_4	
K	D_k	

Ans = ~~min~~ $\max(D_1, D_2, \ldots D_k)$

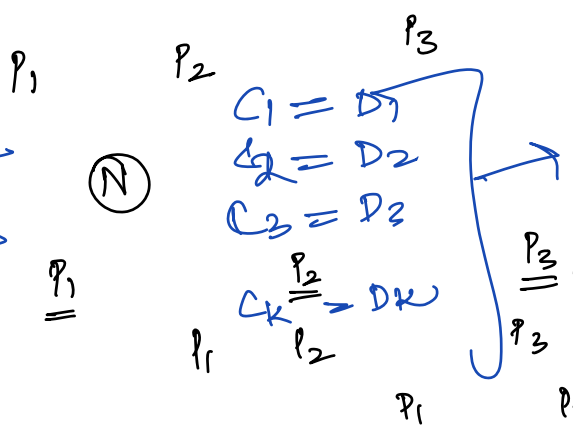


maximum min distance

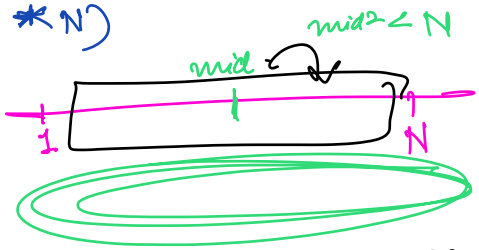
find the best configuration.

↓

min distance b/w any 2 people would be max.



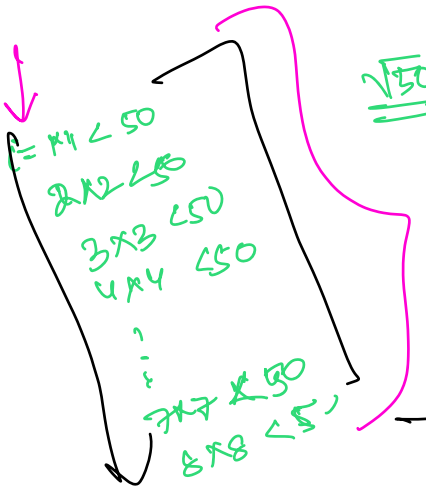
Total config. $O(N_{C_M} * N)$

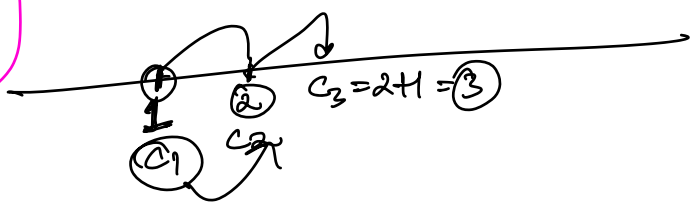


$\sqrt{50}$.

$8^2 > 50$

$mid^2 \not< N$

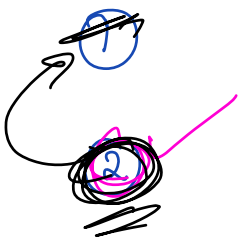


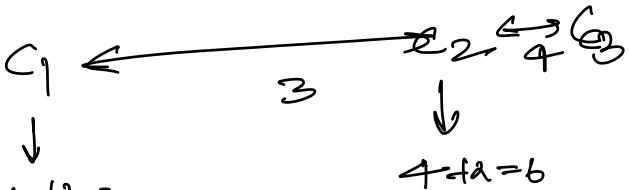


~~K=3~~.

1 2 4 ~~8~~ 9

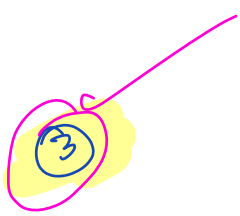
C_1 C_2 C_3





$1+2=3$.

$4+2=6$



C_1

$1+3=4$

C_2 C_3

$4+3=7$

C_1

$1+4=5$

C_2

(8)

$8+4=12$

Search space!

a[0] a[1] a[2] --- a[n-1]

C1 ↑ C2 ↑

max dist → a[n-1] - a[0]

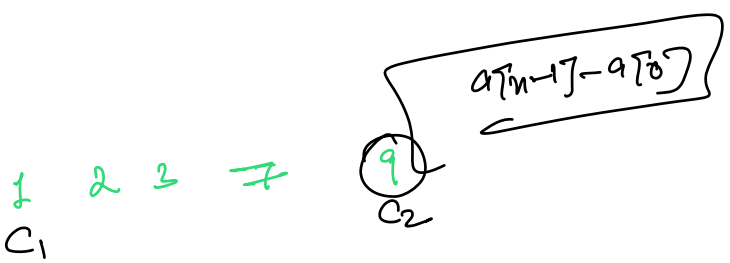
min distance → 1

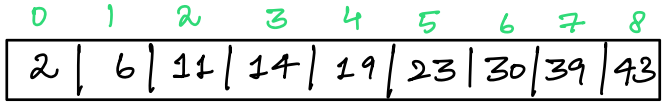
Can I place cows such that distance b/w adjacent cows is atleast 'mid'

yes → l = mid+1.

NO. → Go left. r = mid-1.

C2.

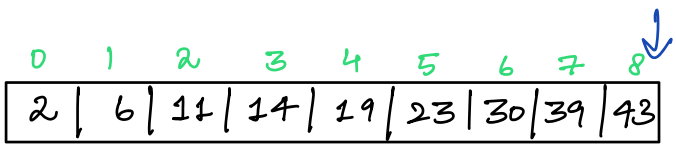




M=4.

l = 1
r = a[n-1] - a[0] = 42.

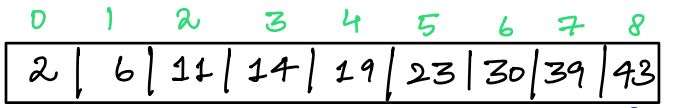
l = 1
r = 42
mid = 21



$C_2 \rightarrow 2+21 = 23$. mid = 21.

$C_3 = 23+21 = 44$

l = 1
r = mid - 1 = 20
mid = 10

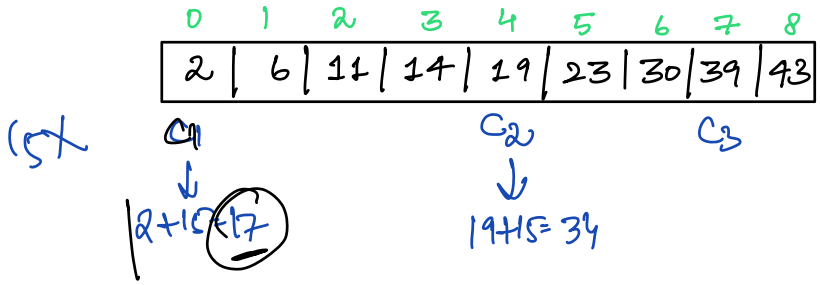


2+10

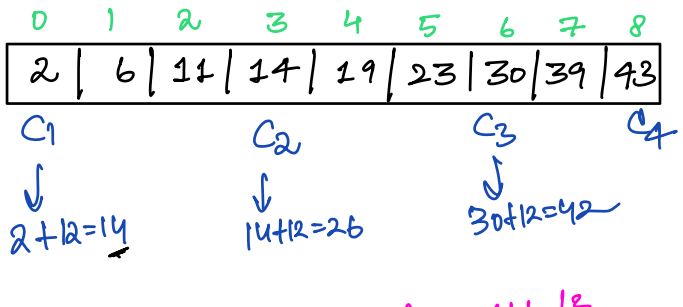
14+10=24

30+10=40

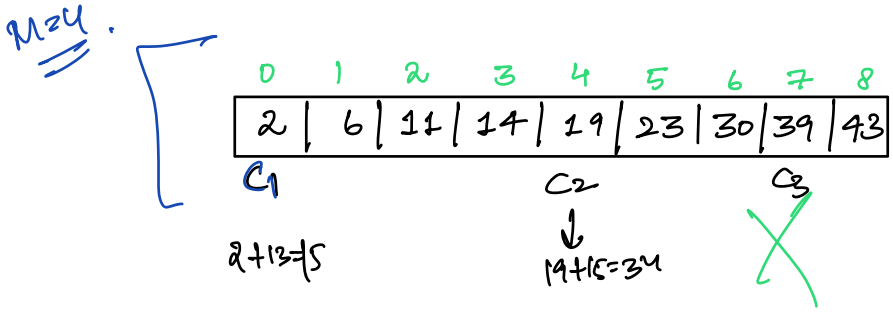
l = mid+1 = 11
r = 20
mid = 15



l = 11
r = mid-1 = 14
mid = 12



l = mid+1 = 13
r = 14
mid = 13.



mid = 14.

l = 13
r = 12

l = 13
r = mid-1 = 12

Pseudo.

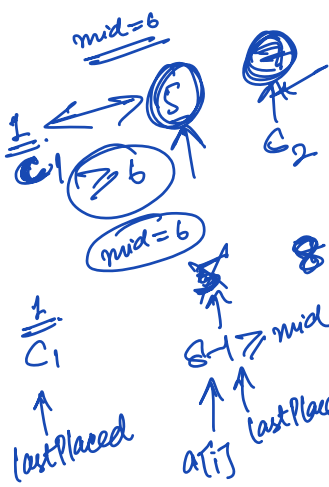
```
int l = 1;
int r = a[n-1] - a[0];
while (l <= r) {
    int mid = (l+r)/2
    if (check(mid)) {
        ans = mid;
        l = mid+1;
    }
    else
        r = mid-1;
}
```

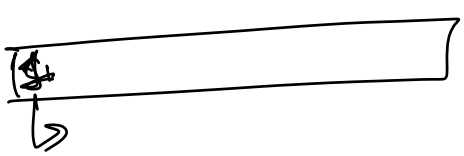
Range = A[n-1] - A[0]

N * log(Range)

```
bool check(int arr[], int mid; int M) {
    int lastPlaced = arr[0];
    int cowsPlaced = 1;
    for (int i=1; i<N; i++) {
        if (arr[i] - lastPlaced >= mid) {
            cowsPlaced++;
            lastPlaced = arr[i];
        }
        if (cowsPlaced == M) return true;
    }
    return false;
}
```

Check T.C.: O(N)





Q. Given N tasks, K workers.

A[i] → time taken to complete i^{th} task.

→ Workers can work parallely.

Constraints:

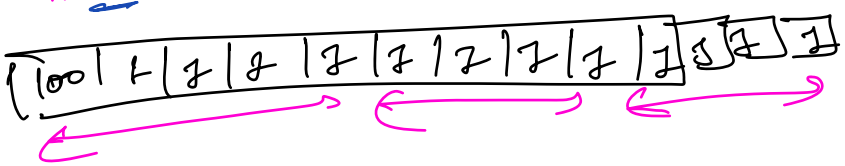
One task can only be done by one person.

A worker can only do continuous tasks.

(5)

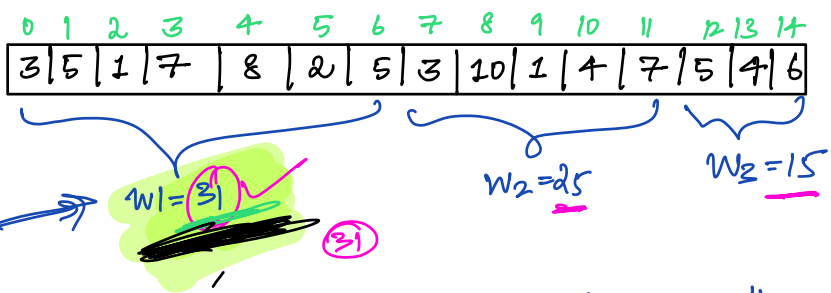
N=15

(3)

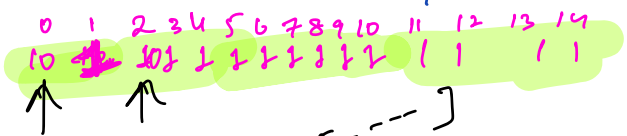


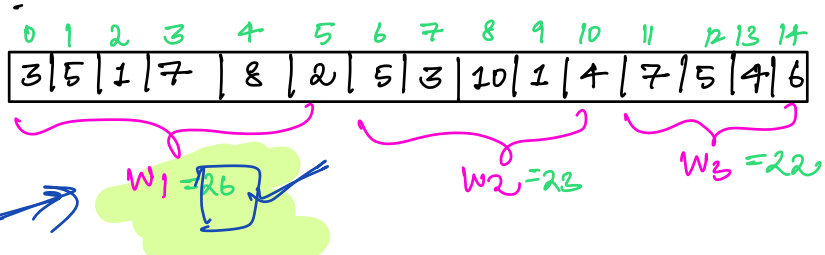
N=15.

K=3.



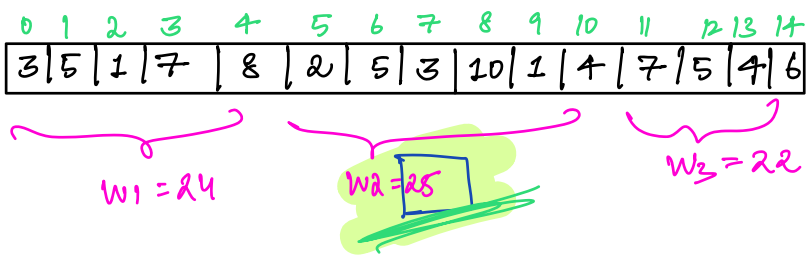
Goal :- We want to decrease the max. time.





mid=25.

[10 11 21 22 23 24 25 26 27 28 29 30]



T= 1 minute
2 minutes
3 minutes
4 minutes
5 minutes

max value (Time) → Sum of array (1 worker does all the jobs)

max(arr)

0	1	2	3
1	2	5	10

min value

N jobs

N workers

N tasks N workers

Frontend 1 day

Backend Job 2 days

Infra Job today

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

l = 10

r = 71

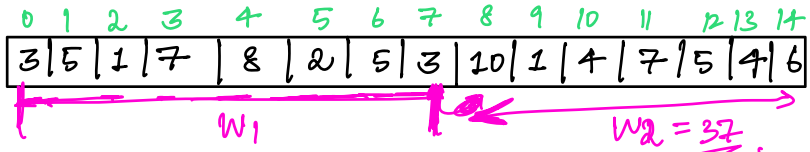
mid = 40.

(what do you check? No worker takes more than 40 minutes)

Yes → Look for a shorter time.

NO → Look for larger times

K=3

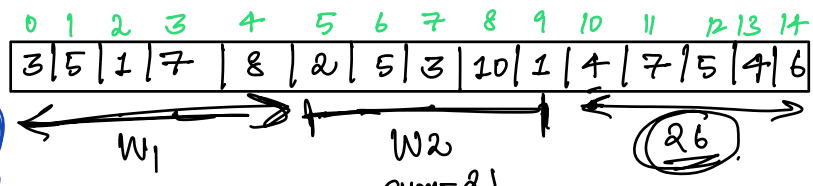


l = 10
r = 71
mid = 40

T.C.

N log(sum) sum = a[0]+a[1]+... a[n-1]

l = 10
r = 39
mid = 24



W1 sum = 21

W2 sum = 21

24 is not a valid answer.

current_time + a[i] > mid

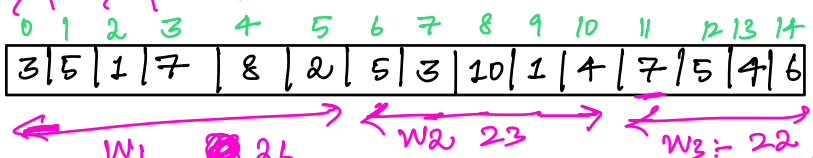
```
bool check(int arr[], int mid, int k){
    int workers = 1;
    int current_time = arr[0]
    for(int i=1; i<n; i++){
        if(current_time + arr[i] > mid){
            workers++
            current_time = arr[i]
        } else {
            current_time += arr[i]
        }
    }
    if(workers <= k) return true
    else return false;
}
```

new worker assigned to this

assign the same worker as before

l = mid+1 = 25
r = 39
mid = 32.

l = 25
r = 31
mid = 28



3, 8, 9, 16,

l = 25
r = mid-1 = 27
mid = 26 ?

Q. You have an array of N +ve integers.
You want to figure out the maximum k such that no subarray of size k has a sum greater than the given sum S.

A = [1, 20, 3, 7] , S = 25

Subarrays of size 1

[1] → 1
[20] → 20
[3] → 3
[7] → 7

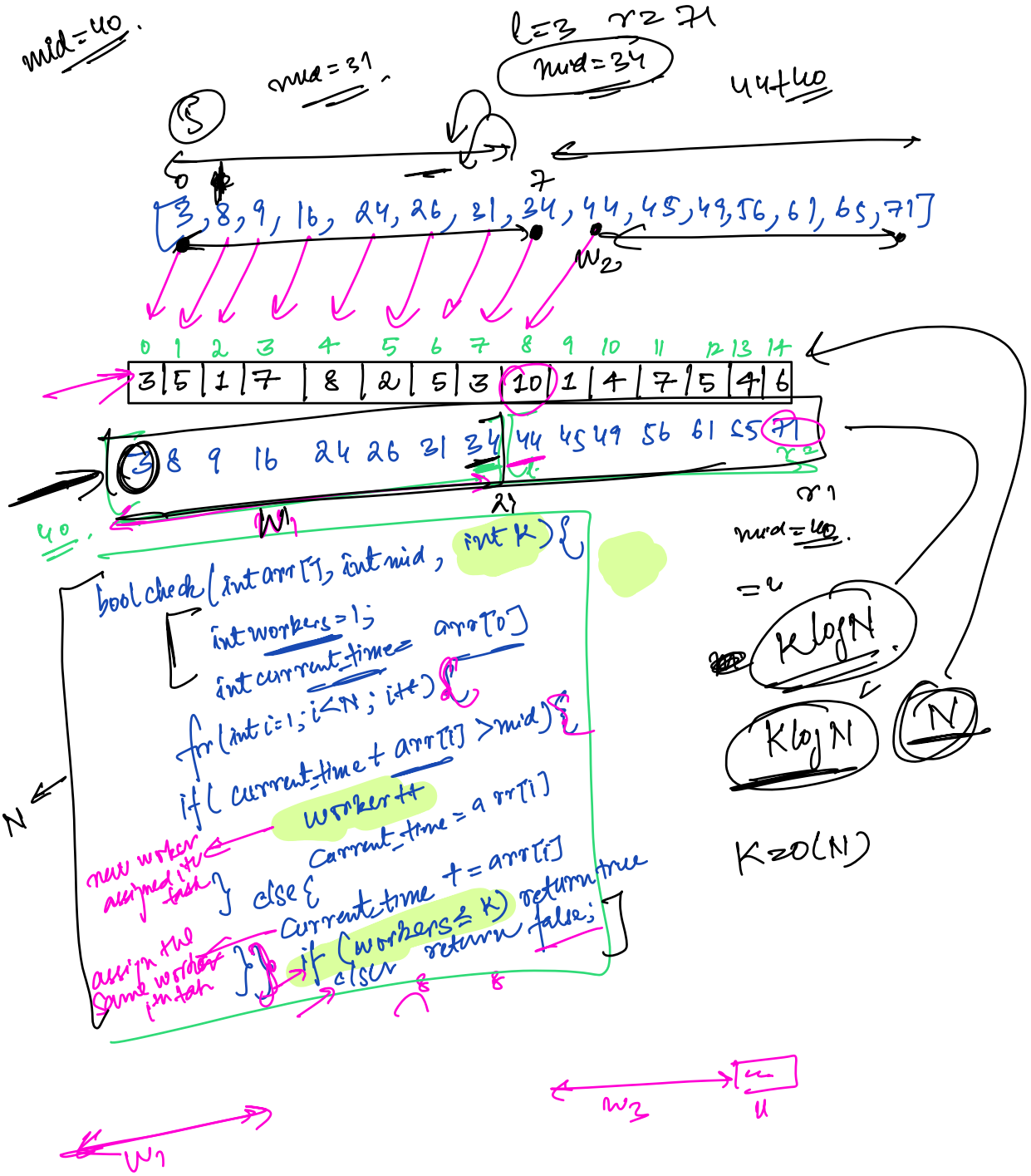
Subarrays of size 2

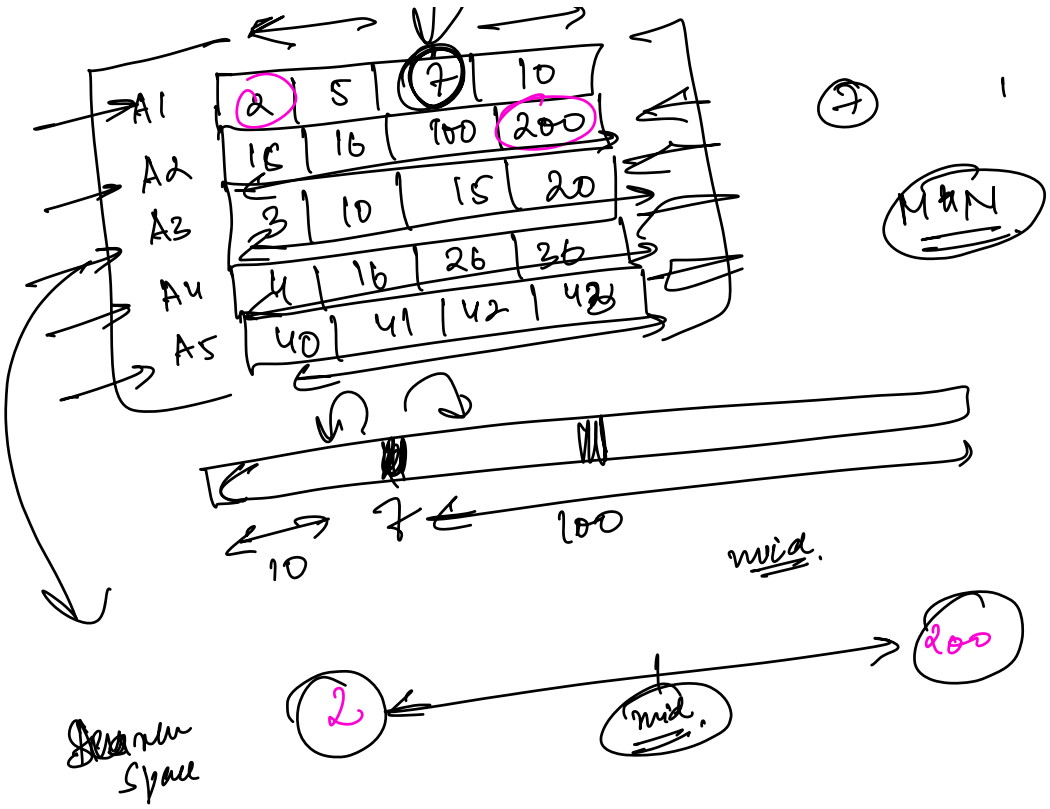
[1, 20] → 21
[20, 3] → 23
[3, 7] → 10

l =

3!

[1, 20, 3] → 24
[20, 3, 7] → 30


mid=40.
mid=31.
l=3 r=71
mid=34
44+40
[3, 8, 9, 16, 24, 26, 31, 34, 44, 45, 49, 56, 61, 65, 71]
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 26 31 34 44 45 49 56 61 65 71
W1
W2
40.
mid=40.
=2
KlogN
KlogN
N
K=O(N)
N
bool check(int arr[], int mid, int K){
int workers=1;
int current_time= arr[0]
for(int i=1; i<N; i++){
if(current_time + arr[i] > mid){
worker++
new worker assigned the task
current_time = arr[i]
} else {
current_time += arr[i]
assign the same worker in task
if (workers <= K) return true
else return false;
W1
W2
u


A1
A2
A3
A4
A5
2 5 7 10
15 16 100 200
3 10 15 20
4 16 26 36
40 41 42 43
7
MAN
10
7
100
mid.
200
2
mid.
Search Space