When to use Binary Search.

1) Target

2) Search Space = [ ]

3) Condition to reduce search by half.

Q1 Given N tasks and K workers and time taken for each task. Find minimum time in which we can complete all tasks.

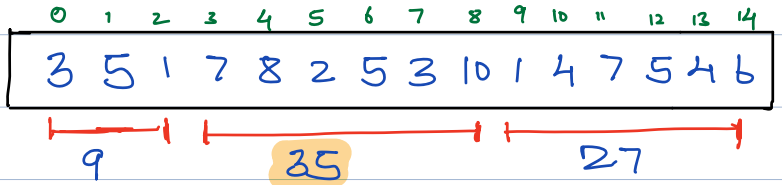
Note 1: A single worker can only do continuous set of tasks.

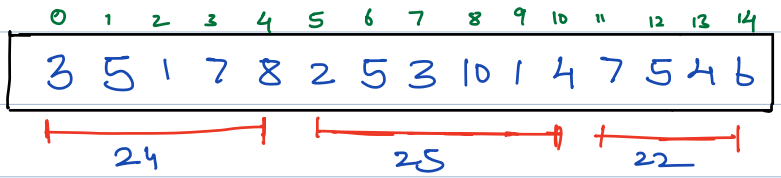
Note 2: All workers start their assigned tasks at the same time.

Note 3. A task can be only assigned to a single user.

Ex1 Total tasks (N) = 15 , No of workers (K) = 3

time_for_task[15] =

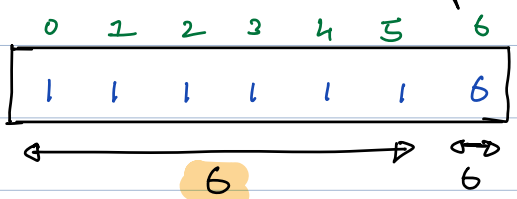




⇒ 25

Ex2 Total tasks (N) = 7 , No of workers (K) = 2

time_for_task[7] =



Ex3 Total tasks (N) = 8 , No of workers (K) = 3

time_for_task[8] =	0	1	2	3	4	5	6	7
10	9	4	5	2	9	10	8

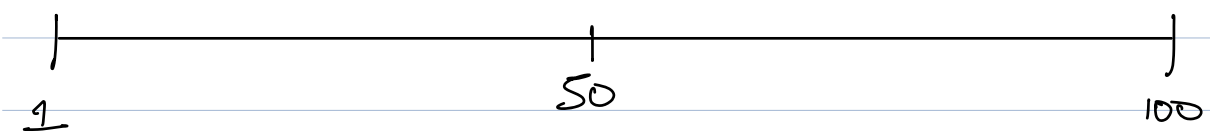
Sum ⇒ 57

Avg ⇒ 19

19 | 20 | 18

Binary Search

1) Target ⇒ minimum time to solve all task
2) Search Space ⇒ [max(Array) , sum(Array)]
3) Do we have a condition to reduce search space.



CASE 1: It is possible to do work in mid time. ⇒ go left Store ans.

CASE 2: It is not possible to do work in mid time. ⇒ go right.

time_for_task[15] =

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

24 25 22

mid = 25

TC: To Run Binary Search

Search Space ⇒ Sum(arr) − max(arr)

⇒ $\log_2$(Search Space) X n

time_for_task[15] =

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

prefix Sum[15]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	8	9	16	24	26	31	34	44	45	49	56	61	65	71

lo ⇒ 10, h1 ⇒ 71, mid ⇒ 40 ✓ h1 = mid-1

lo ⇒ 10, h1 ⇒ 39, mid ⇒ 24 ✗ l ⇒ mid+1

lo ⇒ 25, h1 ⇒ 39, mid ⇒ 32 ✓ ~~l~~ h1 = mid-1

lo ⇒ 25, h1 ⇒ 31, mid ⇒ 28 ✓

lo ⇒ 25, h1 ⇒ 27, mid ⇒ 26 ✓

lo ⇒ 25, h1 ⇒ 25 mid ⇒ 25 ✓

Ex1 18 18 18 3

w1 w2 w3 w4

mid ⇒ 20

K ⇒ 3

⇒ 60

Pseudo Code.

```
bool check ( int time, int arr[], int k, int n ){
int s => 0, c => 0.

for ( int i=0 , i<n ; i++) {
    s => s + arr[i];
    if (s > time) {
        s => arr[i];
        c++;
    }
}
if (s!=0)
    c++;

if ( c > k)
    return false
else
    return true ;

}
```

=> O(n)

2	4	7	8	9

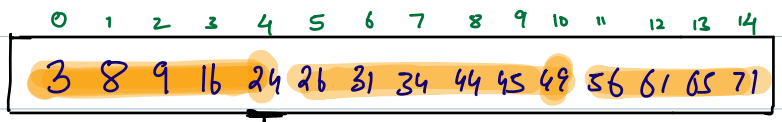
time => 9 (4)

```
int binary_search ( int arr[], int n, int k ) {
    lo => max(array) , int ans => -1
    hi => sum(array).

    while (lo <= hi) {
        int mid = (lo+hi)/2;
        if (check(mid, arr, k, n)) {
            hi => mid-1;
            ans = mid;
        } else {
            lo => mid+1;
        }
    }

    return ans;
}
```

Tc : $O\left(n \log\left\{\text{sum(arr)} - \text{max(arr)}\right\}\right)$

Sc : $O(1)$

Can we do it using prefix sum.



24
+25

49

time > 25

≤ time

TC ⇒ log(n) x 1

Sc: O(n)

Q2 Given N cows and M stalls. They are located on the x-axis, at different locations. Place all N cows such a way that min distance between any 2 cows is maximized.

$M \geq N$

Note 1: In a stall only 1 cow can be present

Note 2: All cows have to be placed.

Note 3. Positions of stall are given in sorted order.

Ex1 Stalls ⇒ 5 Cows ⇒ 3

Stalls[5] =

0	1	2	3	4
1	2	4	8	9

$C_1 \xrightarrow{1} C_2 \xrightarrow{2} C_3$

Stalls[5] =

0	1	2	3	4
1	2	4	8	9

$C_1 \xrightarrow{3} C_3 \xrightarrow{5} C_5$

Ex2 Stalls ⇒ 9 Cows ⇒ 4

Stalls[5] =

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

$C_1 \xrightarrow{12} C_2 \xrightarrow{16} C_3 \xrightarrow{13} C_4$

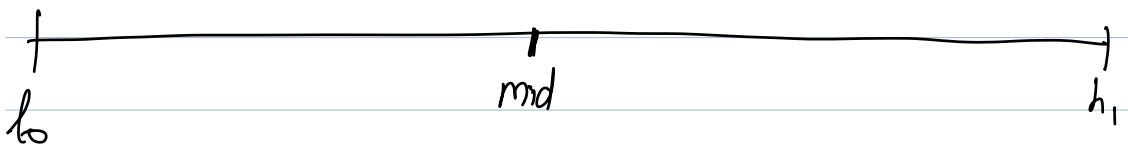
1) target ⇒ minimum distance

2) Search Space ⇒ $[lo, h_1]$

lo ⇒ min diffrence b/w adjacent elements.

3) Condition to reduce search Space.

h_1 ⇒ least — first



Case 1: mid is possible.

ans ⇒ mid. Go right

lo ⇒ mid+1.

Case 2: mid is not possible.

h_1 ⇒ mid - 1

	0	1	2	3	4	5	6	7	8
Stalls[5] =	2	6	11	14	19	25	30	39	43
	C1				C2			C3	

Ans ⇒ 12

lo ⇒ 3 , hi ⇒ 41 , mid ⇒ 22 ✗

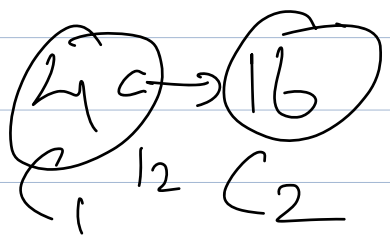
lo ⇒ 3 , hi ⇒ 21 , mid ⇒ 12 ✓

lo ⇒ 13 , hi ⇒ 21 , mid ⇒ 17 ✗

lo ⇒ 13 , hi ⇒ 16 , mid ⇒ 14 ✗

lo ⇒ 13 , hi ⇒ 13 , mid ⇒ 13 ✗

12



12

```
bool check (int distance, int arr[], int n, int c){
    int cnt = 1
    int last_row = arr[0];
    for (int i = 1 ; i<n ; i++) {
        if (arr[i] - last_row >= distance)
            cnt++;
            last_row = arr[i];
            if (cnt == c)
                return true.
    }
    return false;
}
```

no of stalls — no of cows

```
int binary_search (int n, int m, int arr[]) {
```

lo ⇒ min-diffese b/w 2 adjacent elements.

hi ⇒ last - first.

```
int ans = lo / -1 ;
while ( lo <= hi) {
    int mid = ( (lo+hi) / 2 ) ;
    if ( check (mid, arr, n, m))
        ans = mid
        lo = mid + 1.
else
    hi ⇒ mid - 1 .
```

3

Search Space ⇒ hi - lo

Tc : (log (search space) × N)