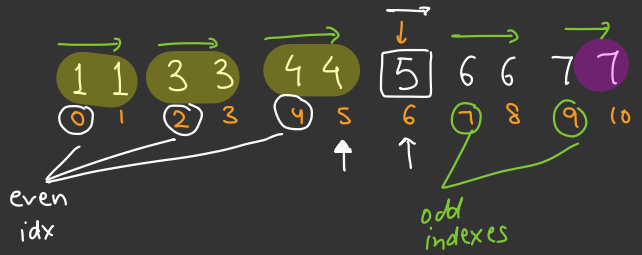
Good evening

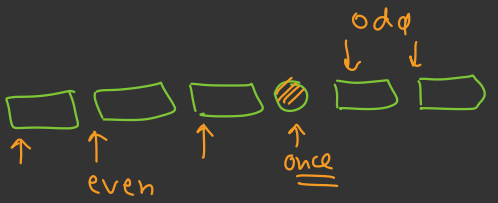
② Sorted Array with 2N + 1 elements, find out unique element.

Repeating, Unique

odd

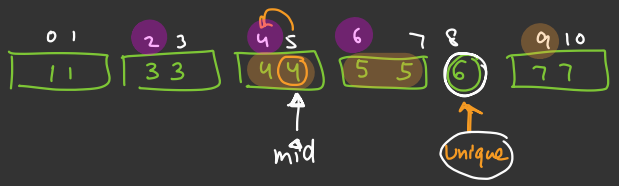


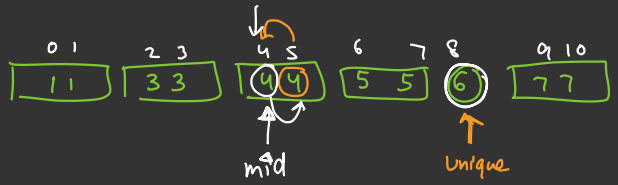
s=0 e=10 mid=5



mid → 6


```
if (a[mid] != a[mid+1]
    a[mid] != a[mid-1]) {
        return a[mid];
}
```





mid = 5

if

mid

odd idx

even

if (a[mid] == a[mid-1])

⇓

Unique element is to Right to mid

if (a[mid] == a[mid+1])

⇓

Unique element is to Right to mid

s = mid + 1

else

{

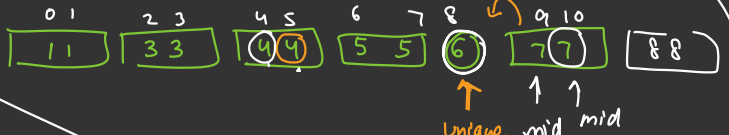
e = mid - 1

}

mid is even
a[mid] != a[mid+1]

mid is odd
a[mid] != a[mid-1]

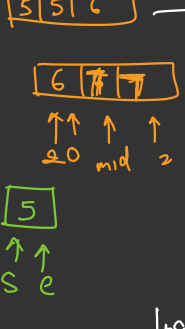
e = mid - 1



Code

```
S = 0
e = arr.length - 1
```

handle unique start / unique end



logic

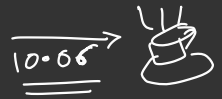
```
// Single size array
if (a[0] != a[1]) { return a[0] }
if (a[n-1] != a[n-2]) return a[n-1];
while (s <= e) {
    mid = (s+e)/2;
    if(a[mid] != a[mid+1] && a[mid] != a[mid-1]) { return a[mid]; }
    else if ((mid is odd && a[mid] == a[mid-1]) || (mid is even && a[mid] == a[mid+1]))
    {
        s = mid + 1;
    }
    else {
        e = mid - 1;
    }
}
```

① Uniq ② Right ② left

fail if unique element is present at start/end of array


?
5
6
6
7
7
8
8
mid

3
3
4
4
5
?
mid



Q2.

N tasks

K workers

Time to finish each task

times for task = [3 (0), 5 (1), 7 (2), 6 (3), 2 (4), 10 (5)]

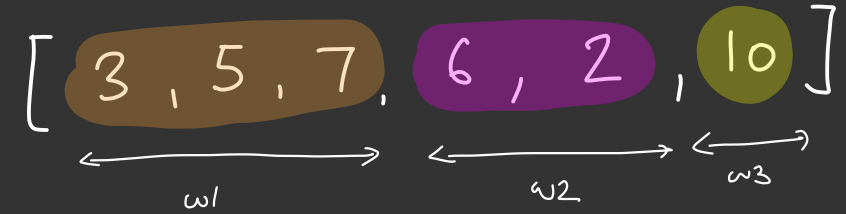
K = 3 workers

Conditions :
- Each worker will execute a "set of continuous tasks".
- All workers start doing tasks at same time

⇒ Find minimum time required to finish all the tasks

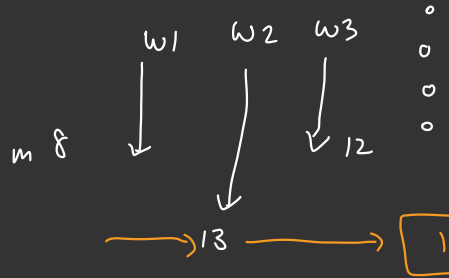
output

min (---max---) optimisation



max (15 mins , 8 mins , 10 mins) = 15 Mins

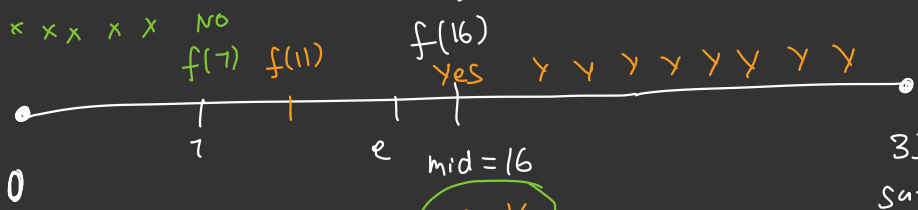
max (8 mins ← 13 mins ← 12 mins →) = 13 mins ✓



Search Space ⇒ Min Time

if all tasks can be finished in 16 mins

↑ every workers should do work ≤ 16 mins



S=6 e=15 mid=7

3 5 7 6 2 10

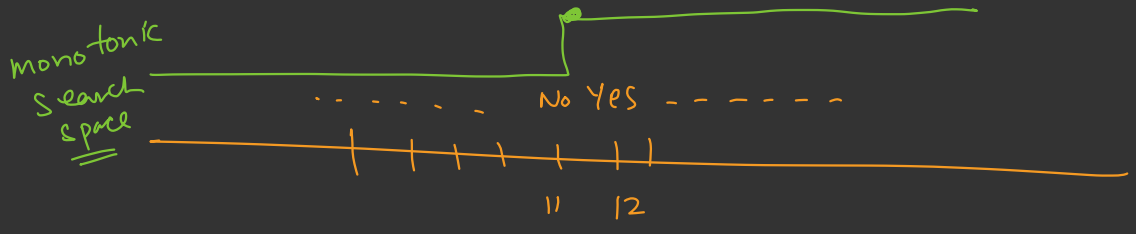
w1 = 3
w2 = 5
w3 = 7

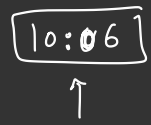
S=8 e=15

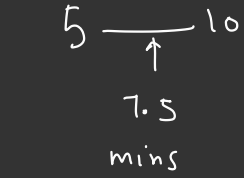
3 5 7 6 2 10 w = 3

min Time

w1 = 3+5+7 ≤ 16
w2 = 6+2 ≤ 16
w3 = 10 ≤ 16







tasks = [3, 5, 7, 6, 2, 10]

k = 3 (workers)

Binary search

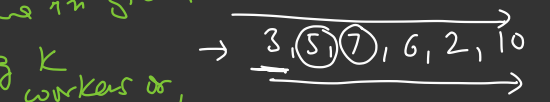

```
s = 0
e = sum of tasks
ans = e
while (s <= e ) {
        mid = (s+e)/2,
        if ( canDo (tasks, mid, k)) {
                ans = mid;
                e = mid - 1,

        else {
```

```
        s = mid + 1;
    }
}
print(ans);
```

mid = 12

```
bool canDo(tasks[], mid, k) {
```

if all task can be done in given by using K workers or not



canDo Method

```
cnt = 1
t = 0
for(i=0; i<=n-1; i++) {
    if(t + time[i] <= mid) {
        t = t + time[i]
    } else {
        t = time[i];   <- Reset time for next worker
        cnt++;
        if(cnt > k) { return false; }
    }
}
```

tasks

cnt = 0 t = 0 + 3 ≤ 12

+ 5 ≤ 12

cnt = 2 t = 7

cnt = 3 t = 6 + 2 ≤ 12

~~8 + 10 ≤ 12~~

cnt = 4 t = 10

cnt = 0 t = 3 ≤ 12

+5

3

return true.

3

mid = 13

3, 5, 7, 6, 2, 10

t = 0

cnt = 1 3 + 5 ≤ 13

cnt = 2 7 + 13 ≤ 13

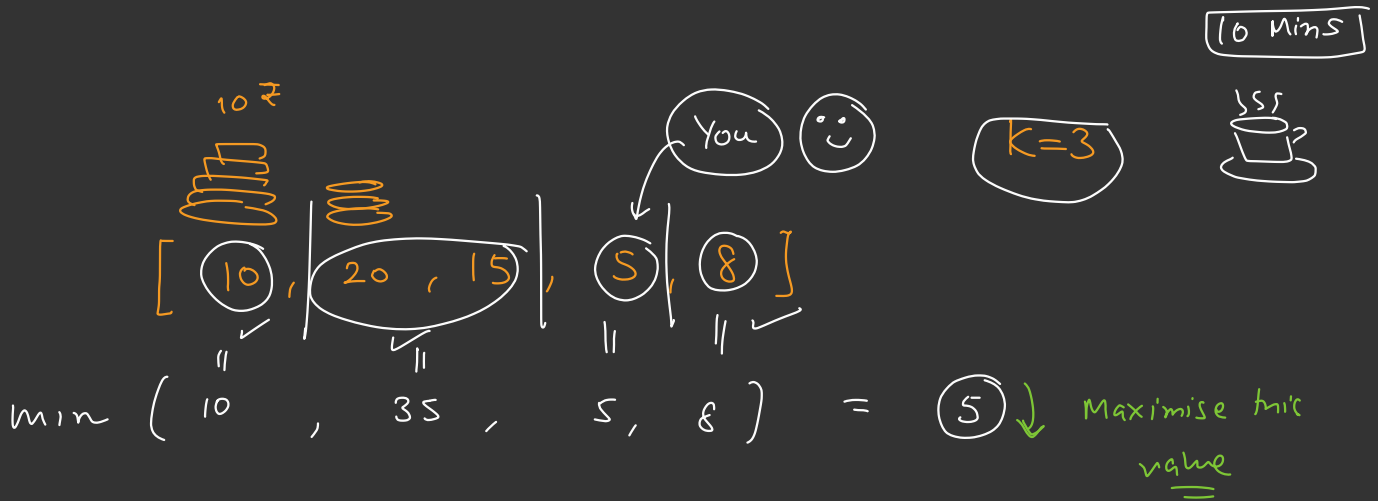
cnt = 3 2 + 10 ≤ 13

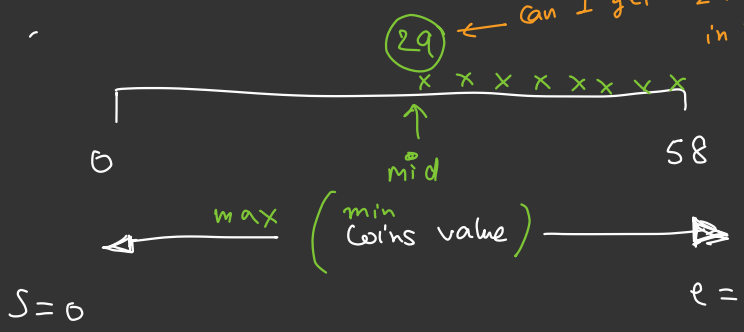
Google Interview Problem

There are N stacks of coins, each stack has certain value of coins.

Consecutive

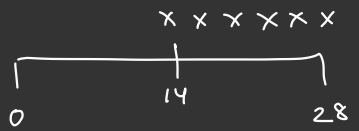
The stacks are lying on a straight line, you have K friends with who you can divide the stacks in K+1 partitions. Your friends are greedy and they will pick the best K parts and you will give the left one. Maximise the value of coins you can get.





Practice Time

10:55



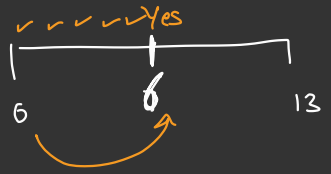
k = 3

k + 1

4 parts

f1 = 10 + 20 = 30

f2 = 15 + 5 + 8 = 28



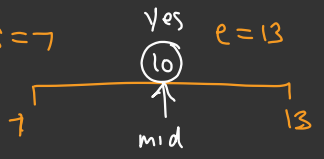
f1 = 10 + 20 = 30

f2 = 15

f3 = 5 + 8 = 13

ans = ~~6~~

ans = 10



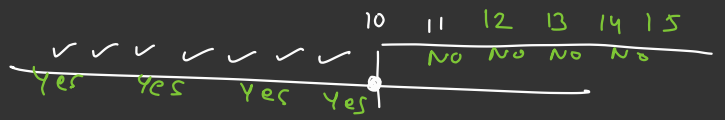
10, 20, 15, 5, 8

f1 = 10 ✓

f2 = 20 ✓

f3 = 15 ✓

f4 = 5 + 8 = 13 ✓

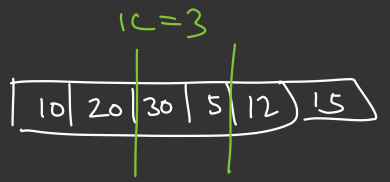


$\vec{10}, \vec{20}, \vec{15}, \vec{5}, \vec{8}$

$f1 = 10$
$f2 = 20$
$f3 = 15$
$f4 = 5 + f = 13$

→ Prata (SPOJ)
→ Painter's Partition Problem]

Binary Search

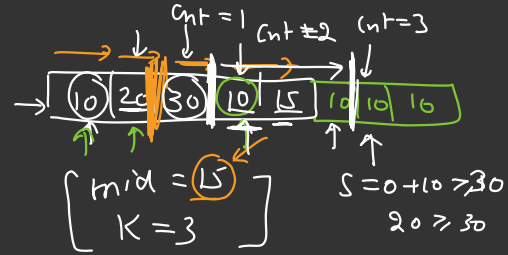



```
s = 0
e = sum of all array elements

                    log[e]
```

```
while (s <= e ){
    mid = (s+e)/2,
                                  O(N)
    if( canDivide( arr, mid, K) {
        ans = mid;
        s = mid + 1
    }
    else {
        e = mid - 1
    }
}
```

$$O\left(N \cdot \log(\text{Array Sum})\right)$$

```
canDivide ( arr, mid, K) {
    cnt = 0        // Partitions
    S = 0
    for(i=0; i<=n-1; i++) {
        if ( S + arr[i] >= mid ){
            cnt++
            S = 0      // Reset sum
        } else {
            S = S + arr[i];
        }
    }
    if (cnt >= K) { return true; }
    else          { return false; }
}
```



→ S = 10+20 = 30 partition

→ S = 30 S >= mid

S = 0+10+15 = 25

mid = 30

mid = 30

→ 0+10 ≥ 30

S = 10+20 ≥ 30

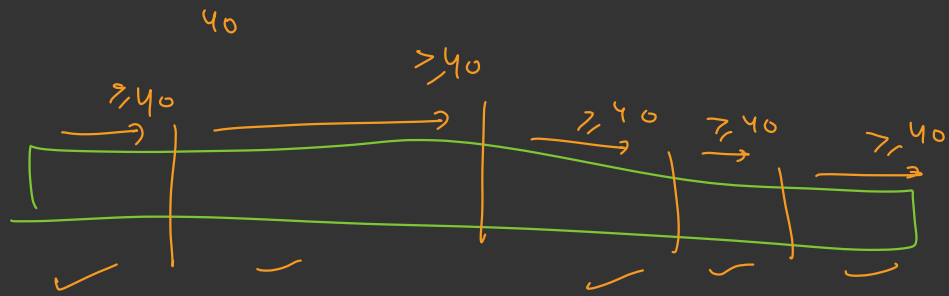
S = 0+30 ≥ 30

S = 0+10 ≥ 30

S = 10+15 ≥ 30

S = 25+10 ≥ 30

S = 0



→ 5 partitions

→ 3 partitions (Yes)