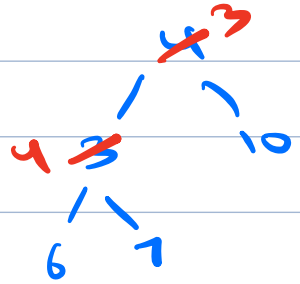
Agenda

- Heap Sort
- Median of stream of integers
- Greedy
- Max no. of jobs

$O(\log_2 N)$ {
- insert ( )
- extractMin ( ) / extractMax ( )
- getMin ( ) / getMax ( )

Heaps → Priority Queue

Contest : Math, 2 pointers, Backtracking, LL and Trees



A = [4 3 10 6 7]

[3 4 10 6 7]

1. Sort the array using heap.

Approach 1 : ① Convert given array into min heap → N

② Call extractmin() N times and store value in ans array.

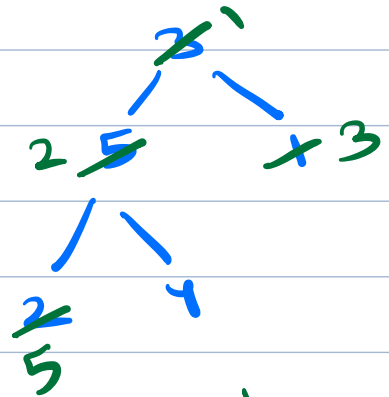
TC : $O(N + N\log N) \approx O(N\log N)$

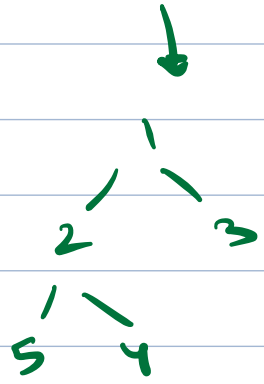
SC : $O(N)$ → ans[]

Can we optimize the SC ?

A = 3 5 1 2 4

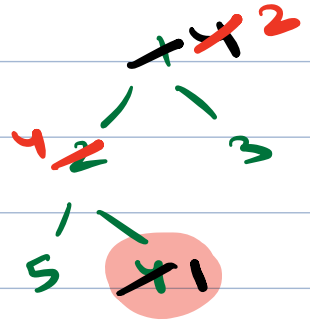
↓ Build heap from array





A : 1 2 3 5 4

① Extract_Min()



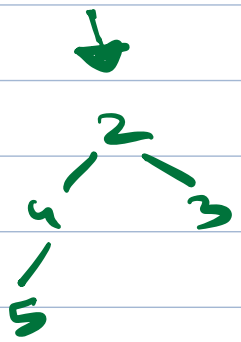
N = 5

swap(A[0], A[4])

N-- → N = 4

heapify(0)

A : 4 2 3 5 1



A : 2 4 3 5 1

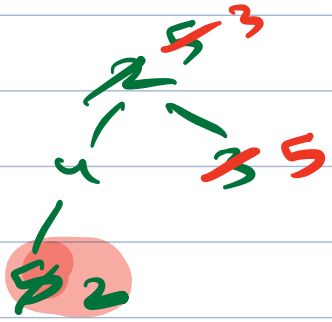
② Extract min

N = 4

swap(A[0], A[3])

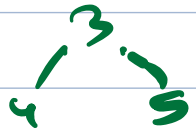
N-- → N = 3

heapify(0)



A : 5 4 3 2 1

A : 3 4 5 2 1



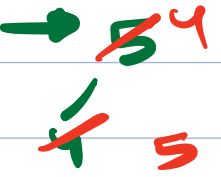
③ Extract Min()

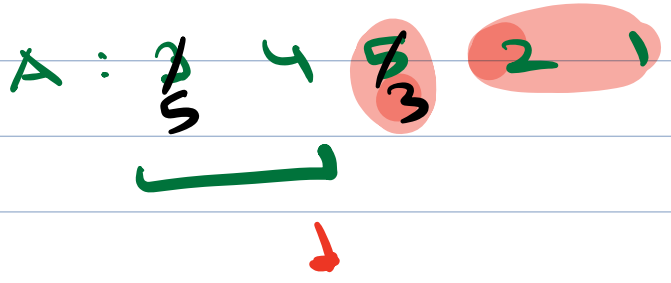
N = 3

swap(A[0], A[2])

N-- → N = 2

heapify(0)





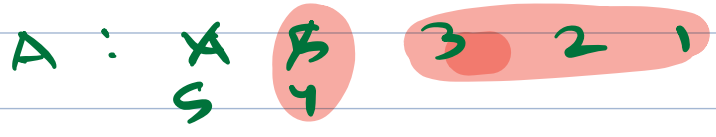
④ Extract Min()

N = 2

swap(A[0], A[1])

N-- ⇒ N = 1

5



① Build min heap from arr

② ExtractMin() N times

③ Reverse arr

TC: O(N log N)

SC: O(1)

OR

① Build max heap
② ExtractMax() N times

TC: O(N log N)
SC: O(1)

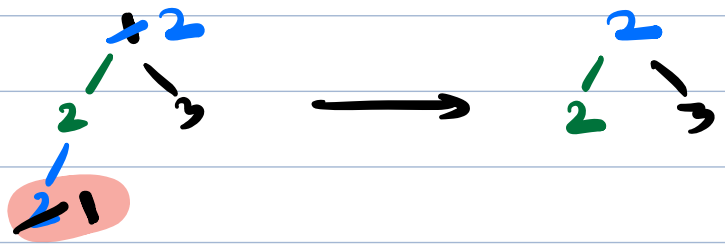
(Delete max virtually from arr)

```
void heapSort(int A[], int N){
    buildMaxHeap(A)
    while(N > 1){
        // extractMin()
        swap(A[0], A[N-1])
        N--
        heapify(0, A, N)   → effective size of heap
    }
}
```

Is heap sort in place ? YES

Is heap sort a stable sorting algo? NO

1 2 3 2 —sort→ 1 2 2 3



2. Given an infinite stream of integers. Find median after every new element comes.

Median → Middle element in sorted data

10 30 15 → 10 15 30

Median

10 30 15 18 → 10 15 18 30

Median = $\frac{15+18}{2} = 16.5$

... 3, 2, 5, 10, 7 → 2, 3, 5, 7, 10

Median = 3, 2.5, 3, 4, 5

Median of [1, 2, 4, 3]

↓ sort

[1, 2, 3, 4]

Median = $\frac{2+3}{2} = 2.5$

3 2

A = [3] A = [3, 2] → A = 2, 3

A = 2, 3, 1 → A = 1, 2, 3

A = 1, 2, 3, 0

Approach 1 : For every new element
→ add to array
→ sort it
→ take middle

1 el → N log N
N el → N^2 log N

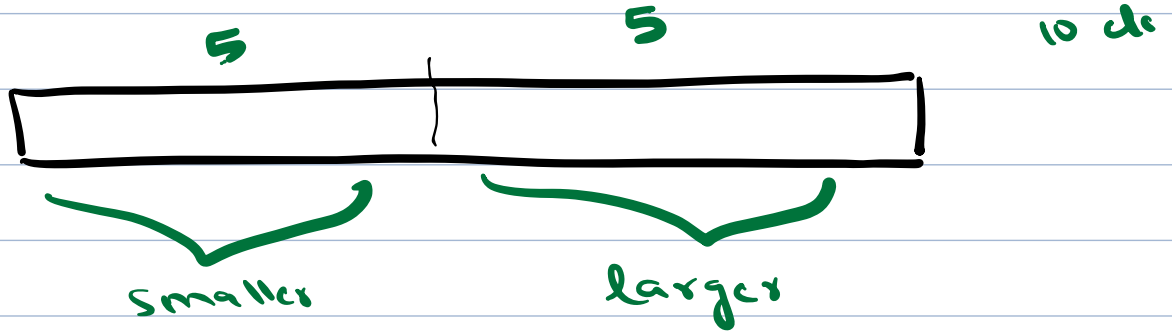
TC : O (N^2 log N)
SC : O (sorting algo)

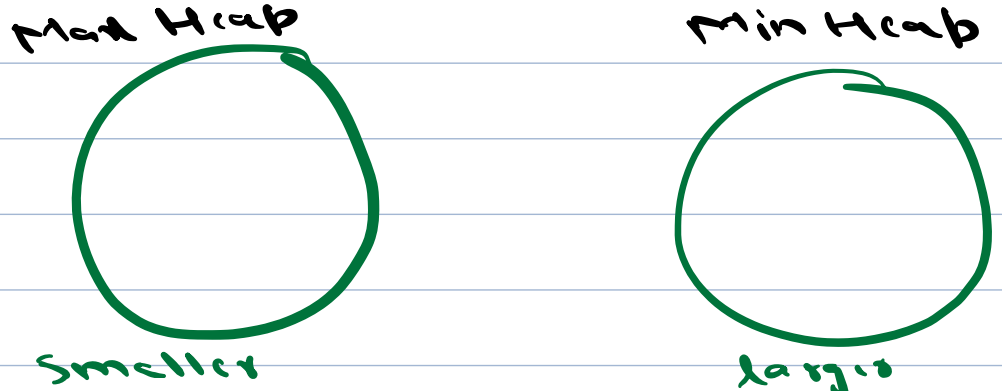
Approach 2 : For every new element
→ add to array and apply insertion sort at it
→ take middle

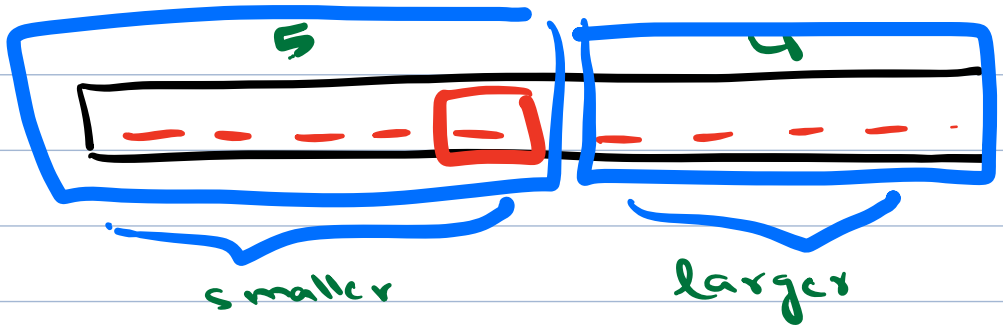
1 el → N
N el → N^2

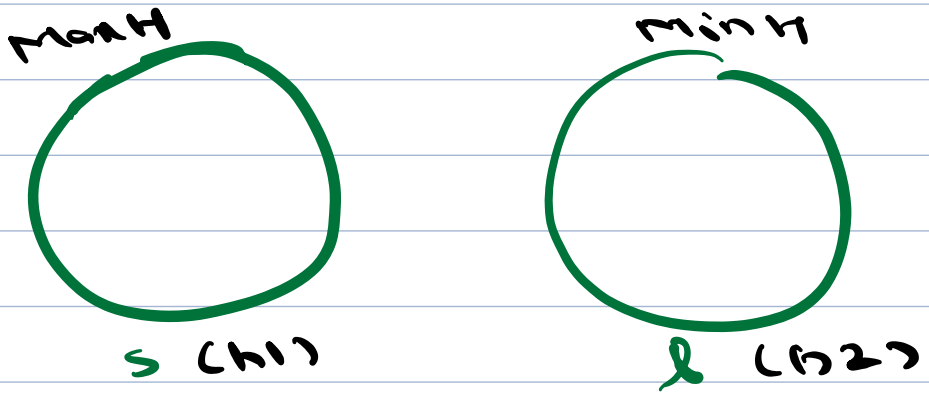
TC : O(N^2)
SC : O(1)

Approach 3 : Heap





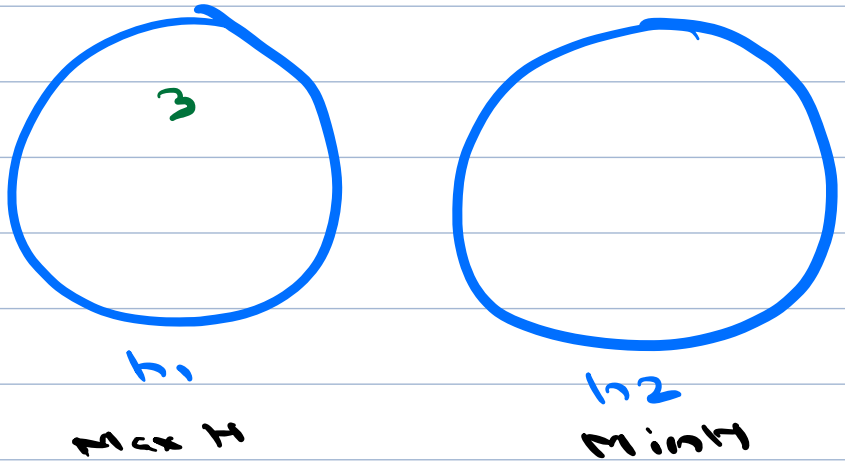


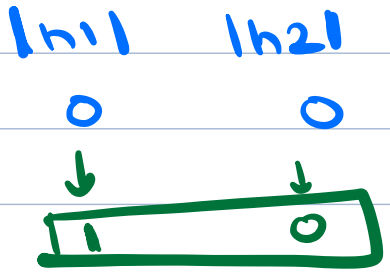


① $h1.size = h2.size$ (Total → even)

② $h1.size - h2.size = 1$ (Total → odd)

① Ele → 3

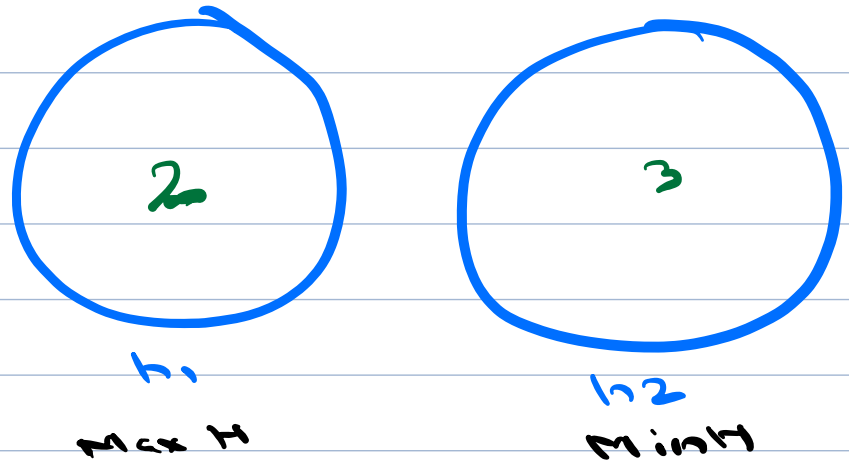


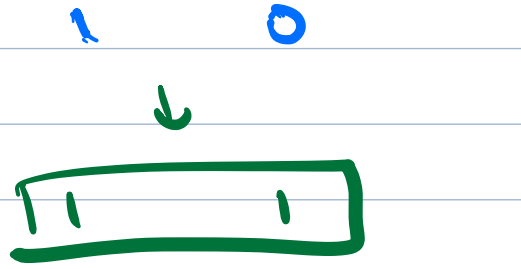


Median = h1.getMax()

ans → 3

② Ele → 2

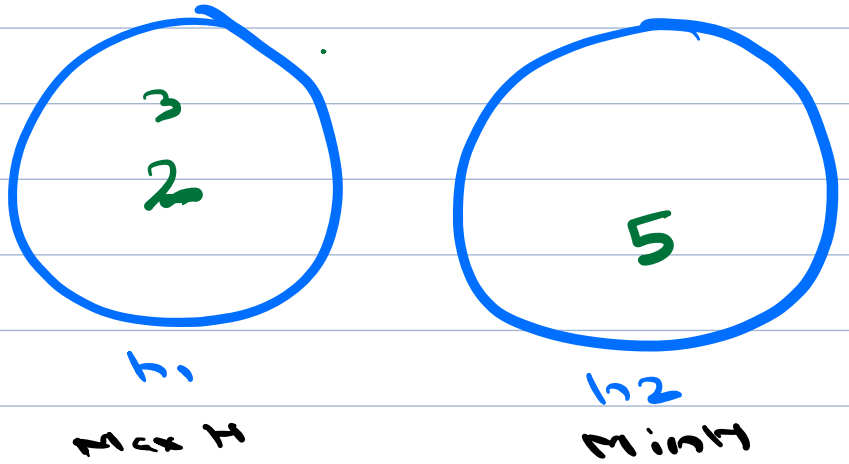


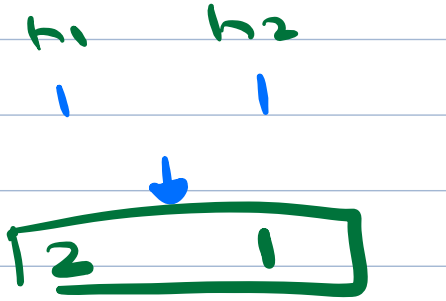


Median = $\frac{2+3}{2}$

$= 2.5$

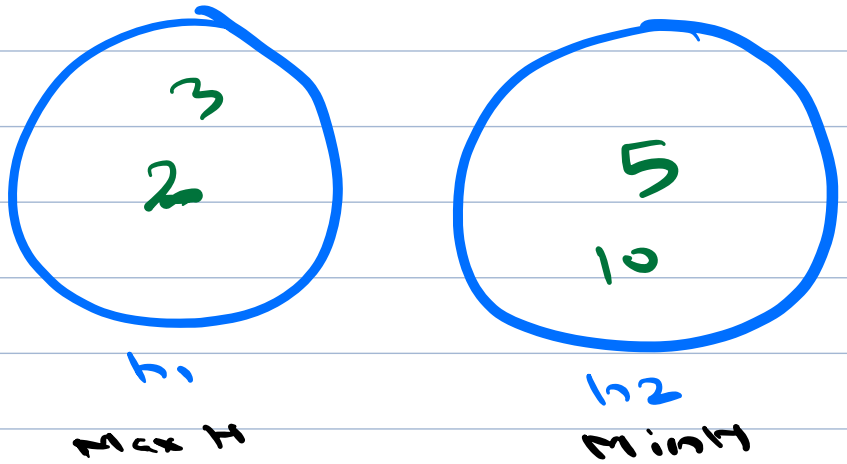
③ Ele → 5

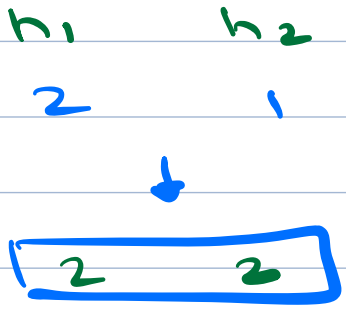




Median → 3

④ Ele → 10

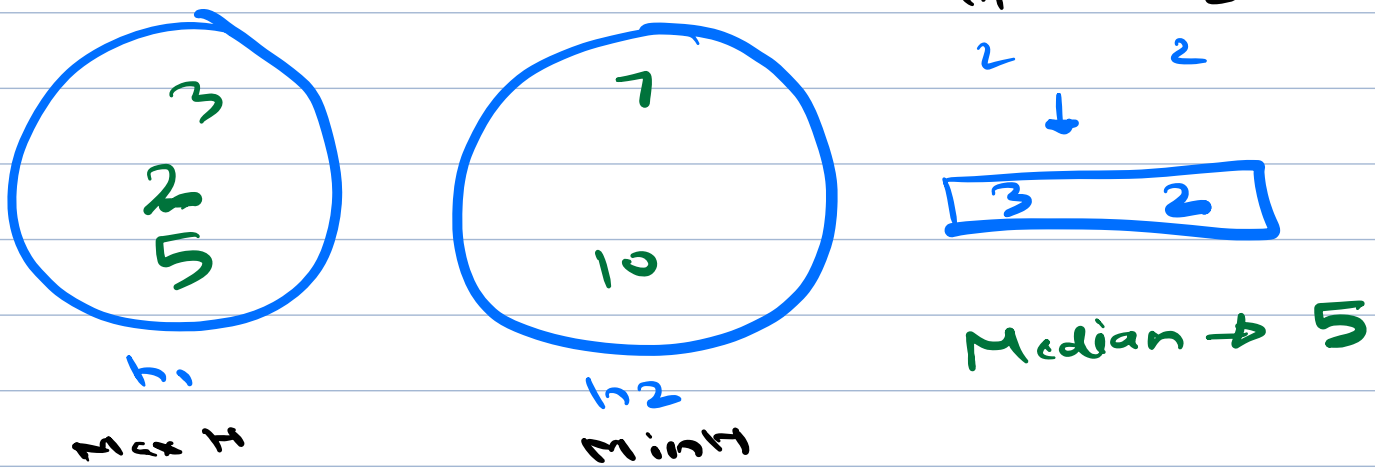


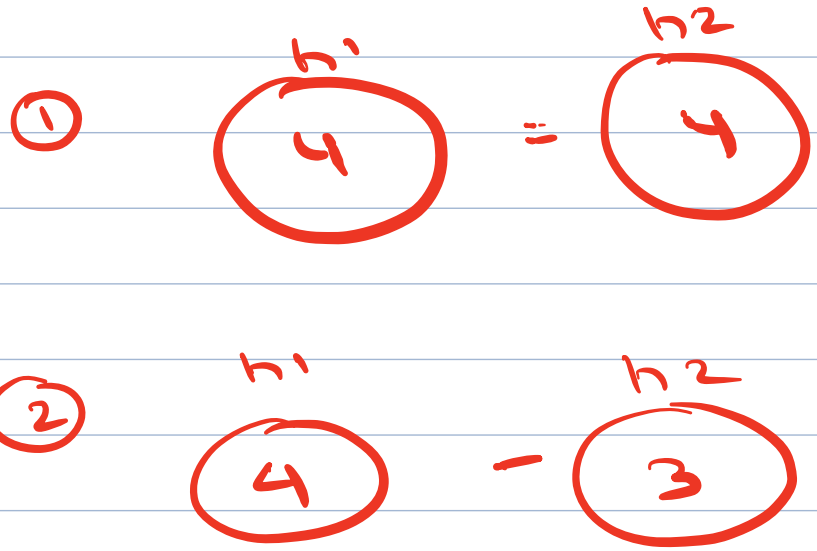


Median → $\frac{3+5}{2}$

$= 4$

(5) Ex → 7





10:40

```
MaxHeap<int>  h1   → smaller
MinHeap<int>  h2   → larger

double getMedian(int x){

    if(h2.size()==0 || x <= h1.getMax()){
        h1.insert(x)
    }
    else{
        h2.insert(x)
    }

    if(h2.size() > h1.size()){
        int temp = h2.getMin()
        h1.insert(temp)
        h2.extractMin()
    }

    if(h1.size() - h2.size() == 2){
        int temp = h1.getMax()
        h2.insert(temp)
        h1.extractMax()
    }

    if(h1.size() > h2.size()){
        return h1.getMax()
```

```
else {
    return ((h1.getMax() + h2.getMin()) / 2.0);
}
}
```

1 insertion → O(log N)

N insertions → O(N log N)

TC : O(N log N)

SC : O(N)

10:40

Greedy Algo

Iphone

Amazon 1.3 l

Flipkart 1.25 l

Paytm 1.35 l

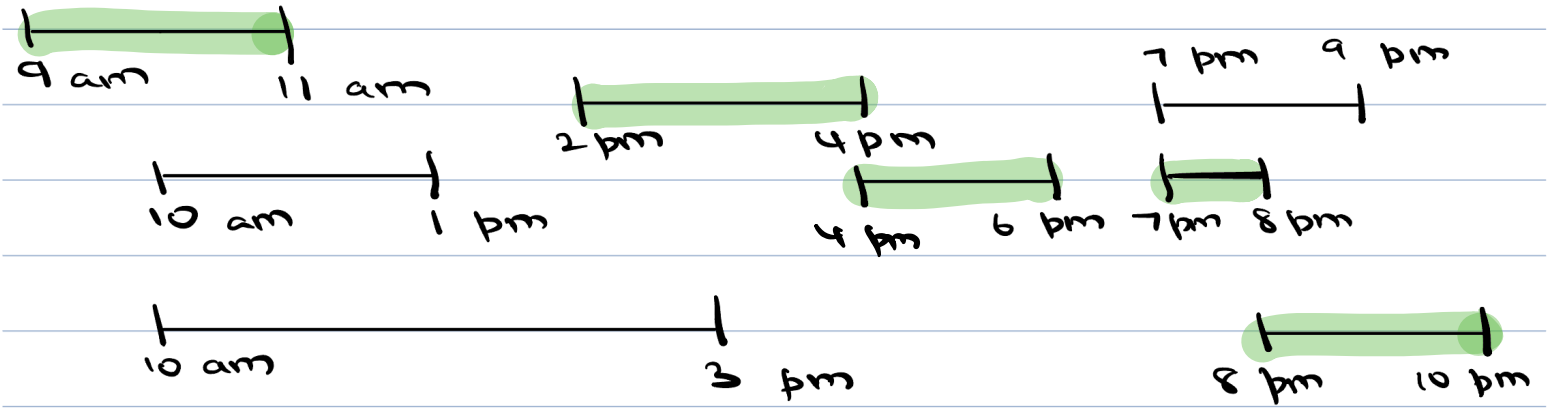
Job

22 lpa

25 lpa

30 lpa

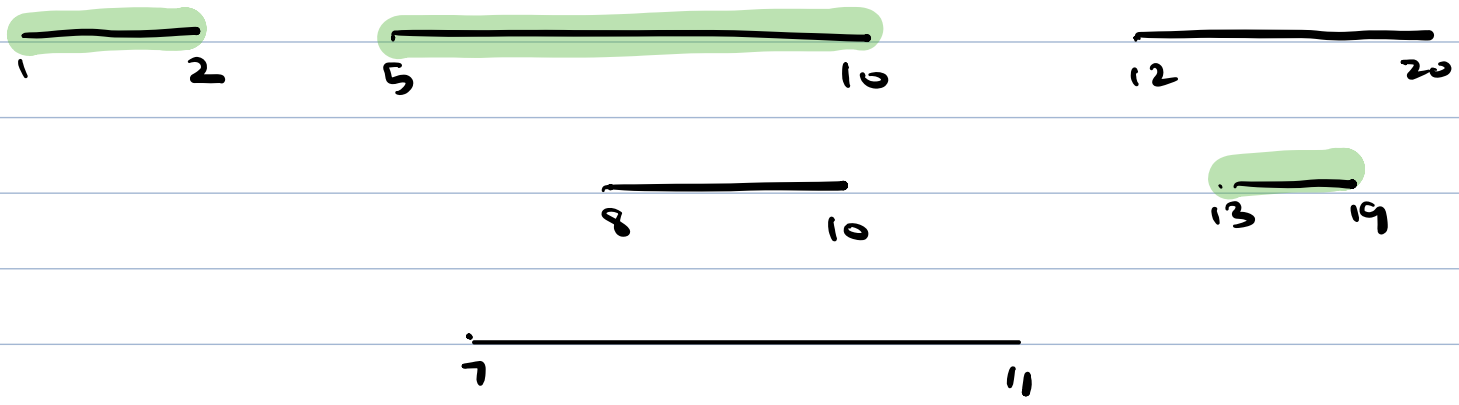
3. Given N jobs with their start & end times. Find max. no. of jobs that can be completed if only one job can be done at a time.



ans = 5

S[nxt] ≥ E[cur]

S = [1 5 8 7 12 13]
E = [2 10 10 11 20 19]



ans = 3

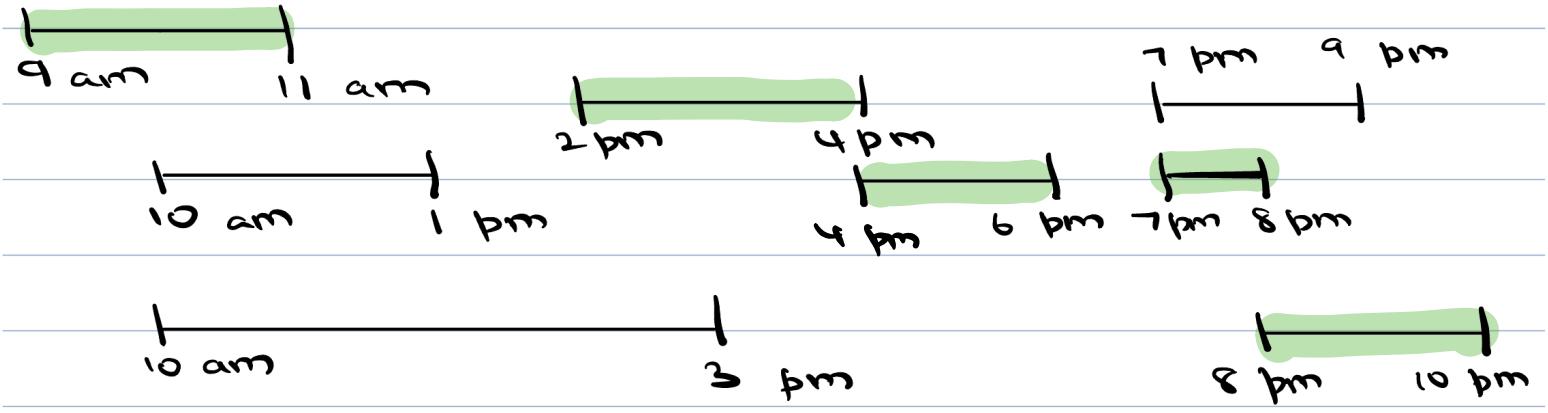
Idea 1 : Sort movies based on duration

Idea 2 : Sort movies based on start time

Idea 3 : Early start time + Less duration

↓

Sort movies based on end time



ans : 5

```
int maxJobs (int S[], int E[], int N) {
    // sort on basis of E[]
    int ans = 0, last = 0
    for (i = 0 ; i < N ; i++) {
        if (S[i] >= last) {
            ans++
            last = E[i]
        }
    }
    return ans
}
```

TC : O(N log N)

SC : O(sorting algo) + array of pairs

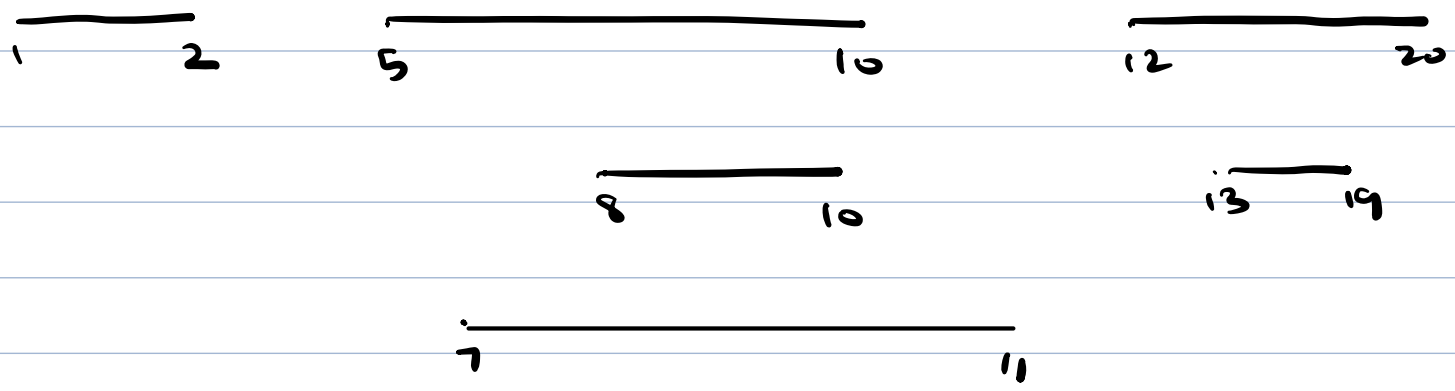
S[], E[]

↓

data (array of pairs)

S = [1 5 8 7 12 13]

E = [2 10 10 11 20 19]



s = [7 13 12 8 5 1

e = [11 19 20 16 10 2]

data = [<7,11> <13,19> <12,20> <8,10>
<5,10> <1,2>]

data = [<1,2> <8,10> <5,10>
<7,11> <13,19> <12,20>]

ans = ~~∅~~ 1 2 3 , last = ~~∅~~ ~~2~~ ~~10~~ 19