Heap, Sort & Greedy

Heap Sort

Given an array, sort it using heap.

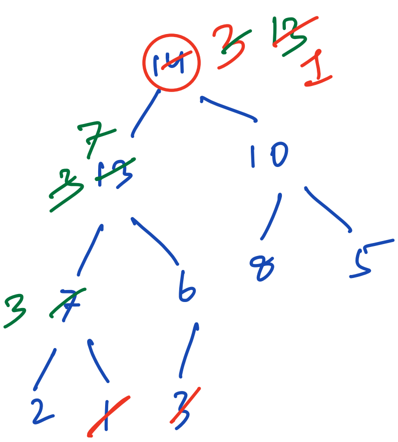
Idea : 1. Build heap → $O(N)$

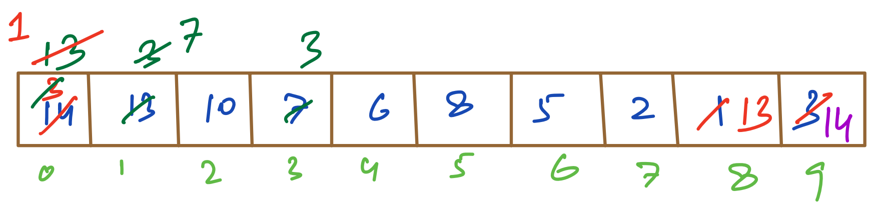
2. getMin() repeatedly & store it in a new array → $O(N \log N)$ $SC = O(N)$

$TC = O(N \log N)$

$SC = O(N) \Rightarrow O(1)$?

Try maxHeap





By doing getmax() repeatedly & replacing it with last element

⇒ heap will become empty & array will be sorted.

Code

// Build max Heap → O(N)

```
j = N-1;
while (j > 0) {
    swap(A[0], A[j])
    j--;
    heapify(A, 0, j) → logN
}
```

↳ also tell the end index

total TC = O(NlogN)

SC = O(1)

Merge Sort   TC = O(NlogN)   SC = O(N)

Quick Sort   TC = O(NlogN) ⇓ O(N²)   SC = O(logN) ⇓ O(N)

Heapsort   TC = O(NlogN)   SC = O(1)

Is heapsort inplace ? ✓ SC = O(1)

Is heapsort stable ? ✗ in heapify order of same values can change

Question

Given an infinite stream of integers.
Find the median of the current set of elements.

Median: middle element in a sorted array

A = [8 5 9] → [5 8 9]   ans = 8

A = [8 5 9 4] → [4 5 8 9]   ans = $\frac{5+8}{2} = 6.5$

A = [1 2 4 3] → [1 2 3 4]   ans = $\frac{2+3}{2} = 2.5$

I/P → 9 8 4 6 7 12 15 . . . . .

O/P → 9 8.5 8 7 7 7.5 8

Bruteforce: for every incoming value, include the value & sort the array & pick the middle.

$$TC = O(N * N\log N)$$

$$= O(N^2 \log N)$$

Insertion sort : every time find the correct position for new element.

$$TC = O(N^2)$$

Use heap

4	5	6

7	8

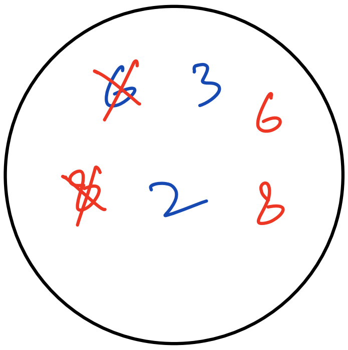
4	5	6	7

8	9	10	12

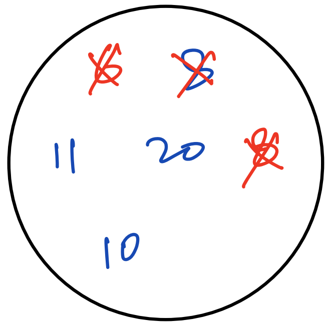
max in 1st half

min in 2nd half

Key is to use maxHeap to store first half & use minHeap to store second half



maxHeap



minHeap

A = 6 3 8 11 20 2 10 . . . . .

o/p → 6 4.5 6 7 8 7 8 . . . .

Code

```
maxH , minH

maxH. insert (A[0])

print( A[0])

for ( i = 1   to   N-1 ) {

    if ( A[i]  <=   maxH. top() ) {

        maxH. insert (A[i])    →

    }
    else {

        minH. insert (A[i])    →
```

O(logN)

```
    }
    int size_diff = maxH.size() - minH.size();

    if( size_diff > 1) {
        minH.insert( maxH.getMax());     -> O(logN)
                                         ↳ move from maxHeap
    }                                      to minHeap

    else if( size_diff < 0 ) {
        maxH.insert( minH.getMin());     -> O(logN)
                                         ↳ move from minHeap
    }                                      to maxHeap


    if( maxH.size() == minH.size() ){
        print( maxH.top() + minH.top() / 2.0);

    }
    else {
        print( maxH.top());
    }
}
```

TC = O(N logN)

SC = O(N)

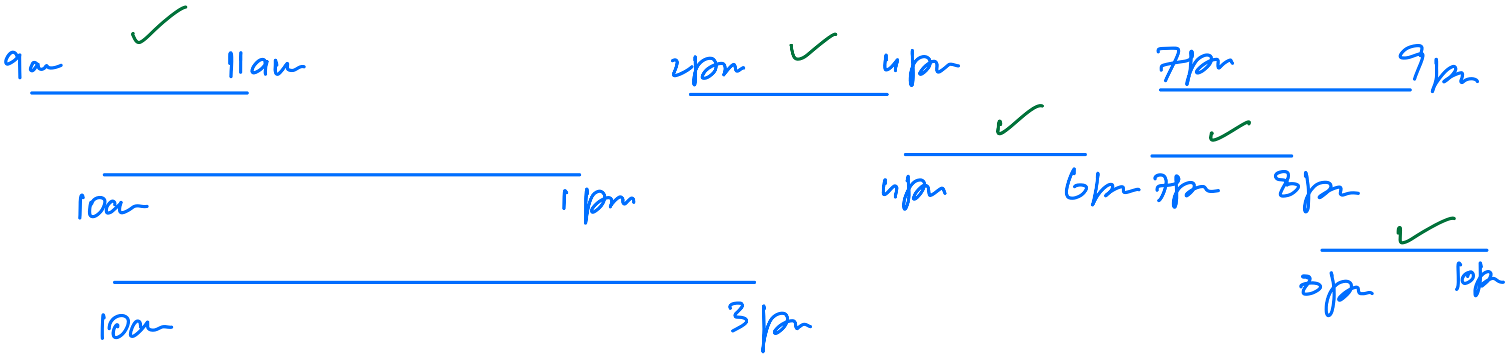
Greedy

Greedy approach deals with maximizing our profit & minimizing loss.

iphone → Amazon (1.3L)
iphone → Flipkart (1.25L)
iphone → Paytm (1.23L)

Company switch → 22 LPA
Company switch → 25 LPA
Company switch → 28 LPA

Given N jobs with their start & end times. Find max. number of jobs that can be completed if only 1 job can be done at a time.

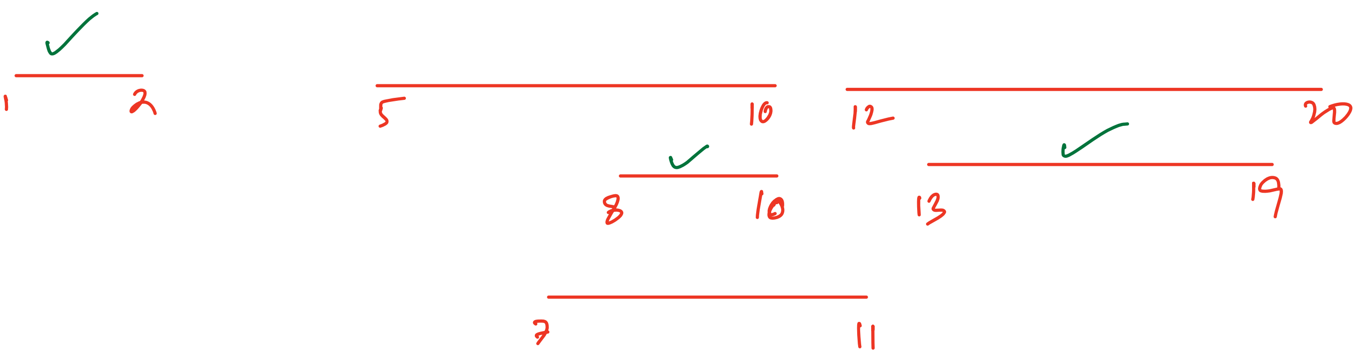


We have to select job whose start time >= end time of previous job.

$$S = \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 5 & 8 & 7 & 12 & 13 \end{array}$$

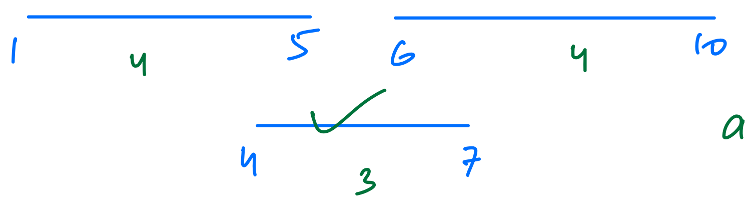
$$E = [\ 2 \quad 10 \quad 10 \quad 11 \quad 20 \quad 19\ ]$$

ans = 3



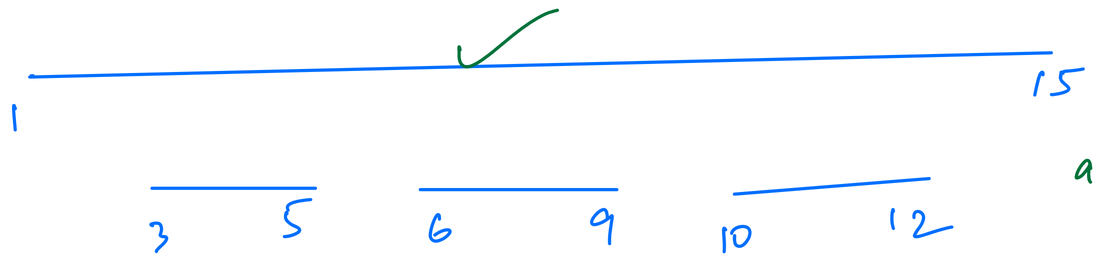
Max # Jobs

1. Pick job with shortest duration first. ✗



ans=1 ✗

2. Pick job with earliest start time. ✗



ans=1 ✗

we want job which start early & have short duration. ⇒ end early

3. Pick job with early end time.

THIS WORKS !!

code

// sort on end time

```
ans=1 , end= E[0]

for(i=1 to N-1){
    if( S[i] >= end){
        ans++
        end = E[i]
    }
}
return
```

	0	1	2	3	4	5
S =	1	5	8	7	13	12
E =	2	10	10	11	19	20

$TC = O(N\log N)$

$SC = O(1)$