



Solution Review: Problem Challenge 2

We'll cover the following ^

- Maximum CPU Load (hard)
- Solution



- Space complexity

Maximum CPU Load (hard)#

We are given a list of Jobs. Each job has a Start time, an End time, and a CPU load when it is running. Our goal is to find the **maximum CPU load** at any time if all the **jobs are running on the machine**.

Example 1:

Jobs: [[1,4,3], [2,5,4], [7,9,6]]

Output: 7

Explanation: Since [1,4,3] and [2,5,4] overlap, their maximum CPU load (3+4=7) will be when both the jobs are running at the same time i.e., during the time interval (2,4).

Example 2:



Jobs: [[6,7,10], [2,4,11], [8,12,15]]

Output: 15

Explanation: None of the jobs overlap, therefore we will take the maximum load of any job which is 15.



Example 3:

Jobs: [[1,4,2], [2,4,1], [3,6,5]]

Output: 8

Explanation: Maximum CPU load will be 8 as all jobs overlap during the time interval [3,4].

Solution#

The problem follows the [Merge Intervals](#) pattern and can easily be converted to [Minimum Meeting Rooms](#). Similar to 'Minimum Meeting Rooms' where we were trying to find the maximum number of meetings happening at any time, for 'Maximum CPU Load' we need to find the maximum number of jobs running at any time. We will need to keep a running count of the maximum CPU load at any time to find the overall maximum load.

Code#

Here is what our algorithm will look like:

 Python3



```
7     self.end = end
8     self.cpu_load = cpu_load
9
10    def __lt__(self, other):
11        # min heap based on job.end
12        return self.end < other.end
13
14
```



```
15 def find_max_cpu_load(jobs):
16     # sort the jobs by start time
17     jobs.sort(key=lambda x: x.start)
18     max_cpu_load, current_cpu_load = 0, 0
19     min_heap = []
20
21     for j in jobs:
22         # remove all the jobs that have ended
23         while(len(min_heap) > 0 and j.start >= min_heap[0].end):
24             current_cpu_load -= min_heap[0].cpu_load
25             heappop(min_heap)
26         # add the current job into min_heap
27         heappush(min_heap, j)
28         current_cpu_load += j.cpu_load
29         max_cpu_load = max(max_cpu_load, current_cpu_load)
30     return max_cpu_load
31
32
33 def main():
34     print("Maximum CPU load at any time: " + str(find_max_cpu_load([job(1, 4, 3), job(
35     print("Maximum CPU load at any time: " + str(find_max_cpu_load([job(6, 7, 10), job(
36     print("Maximum CPU load at any time: " + str(find_max_cpu_load([job(1, 4, 2), job(
37
```



Time complexity#

The time complexity of the above algorithm is $O(N * \log N)$, where 'N' is the total number of jobs. This is due to the sorting that we did in the beginning. Also, while iterating the jobs, we might need to poll/offer jobs to the priority queue. Each of these operations can take $O(\log N)$. Overall our algorithm will take $O(N \log N)$.

Space complexity#

The space complexity of the above algorithm will be $O(N)$, which is required for sorting. Also, in the worst case, we have to insert all the jobs into the priority queue (when all jobs overlap) which will also take $O(N)$ space. The overall space complexity of our algorithm is $O(N)$.


 **Back**

Problem Challenge 2



Problem Challenge 3

 Mark as Completed

 Report an Issue

