Course Schedule III

There are `n` different online courses numbered from `1` to `n`. Each course has some duration(course length) `t` and closed on `d_th` day. A course should be taken **continuously** for `t` days and must be finished before or on the `d_th` day. You will start at the `1_st` day.

Given `n` online courses represented by pairs `(t,d)`, your task is to find the maximal number of courses that can be taken.

Example:

```
Input: [[100, 200], [200, 1300], [1000, 1250], [2000, 3200]]
Output: 3
Explanation:
There're totally 4 courses, but you can take 3 courses at most:
First, take the 1st course, it costs 100 days so you will finish it on the 100th day,
 and ready to take the next course on the 101st day.
Second, take the 3rd course, it costs 1000 days so you will finish it on the 1100th d
ay, and ready to take the next course on the 1101st day.
Third, take the 2nd course, it costs 200 days so you will finish it on the 1300th day
.
The 4th course cannot be taken now, since you will finish it on the 3300th day, which
 exceeds the closed date.
```

Note:

1. The integer 1
2. You can't take two courses simultaneously.

Solution 1

Sort all the courses by their ending time. When considering the first K courses, they all end before `end`. A necessary and sufficient condition for our schedule to be valid, is that (for all K), the courses we choose to take within the first K of them, have total duration less than `end`.

For each K, we will greedily remove the largest-length course until the total duration `start` is `<= end`. To select these largest-length courses, we will use a max heap. `start` will maintain the loop invariant that it is the sum of the lengths of the courses we have currently taken.

Clearly, this greedy choice makes the number of courses used maximal for each K. When considering potential future K, there's never a case where we preferred having a longer course to a shorter one, so indeed our greedy choice dominates all other candidates.

```
def scheduleCourse(self, A):
    pq = []
    start = 0
    for t, end in sorted(A, key = lambda (t, end): end):
        start += t
        heapq.heappush(pq, -t)
        while start > end:
            start += heapq.heappop(pq)
    return len(pq)
```

(With thanks to @uwi)

written by awice original link here

Solution 2

```
public class Solution {
    public int scheduleCourse(int[][] courses) {
        Arrays.sort(courses,(a,b)->a[1]-b[1]); //Sort the courses by their deadlines (Greedy! We have to deal with courses with early deadlines first)
        PriorityQueue<Integer> pq=new PriorityQueue<>((a,b)->b-a);
        int time=0;
        for (int[] c:courses)
        {
            time+=c[0]; // add current course to a priority queue
            pq.add(c[0]);
            if (time>c[1]) time-=pq.poll(); //If time exceeds, drop the previous course which costs the most time. (That must be the best choice!)
        }
        return pq.size();
    }
}
```

written by KakaHiguain original link here

Solution 3

First, we sort courses by the end date, this way, when we're iterating through the courses, we can switch out any previous course with the current one without worrying about end date.

Next, we iterate through each course, if we have enough days, we'll add it to our multiset. If we don't have enough days, then we can either ignore this course, or we can use it to replace a longer course we added earlier.

```
class Solution {
public:
    int scheduleCourse(vector<vector<int>>& courses) {
        // sort courses by the end date
        sort(courses.begin(),courses.end(),
            [](vector<int> a, vector<int> b){return a.back()<b.back();});

        multiset<int> cls; // store lengths of each course we take (could be duplic
ates!)
        int cursum=0;

        for (int i=0; i<courses.size(); i++) {

            // if we have enough time, we will take this course
            if (cursum+courses[i].front()<=courses[i].back()) {
                cls.insert(courses[i].front());
                cursum+=courses[i].front();
            } else if (*cls.rbegin()>courses[i].front()) {
                // if we don't have enough time, we switch out a longer course
                cursum+=courses[i].front()-*cls.rbegin();
                cls.erase(--cls.end());
                cls.insert(courses[i].front());
            } // if we don't have enough time for course[i],
              //and it's longer than any courses taken, then we ignore it
        }

        return cls.size();
    }
};
```

My final consideration was when we replace a longer course with a much shorter one, does that mean we'll have enough room to take some courses previously ignored for being too long?

The answer is no, because any courses we missed would be longer than what's in multiset `cls`. So the increase in number of days cannot be larger than the largest element in `cls`, and certainly will be less than a previously ignored course which has to be even longer.

written by baselRus original link here