6306. Time to Cross a Bridge

There are `k` workers who want to move `n` boxes from an old warehouse to a new one. You are given the two integers `n` and `k`, and a 2D integer array `time` of size `k x 4` where `time[i] = [leftToRight_i, pickOld_i, rightToLeft_i, putNew_i]`.

The warehouses are separated by a river and connected by a bridge. The old warehouse is on the right bank of the river, and the new warehouse is on the left bank of the river. Initially, all `k` workers are waiting on the left side of the bridge. To move the boxes, the ith worker (**0-indexed**) can :

- Cross the bridge from the left bank (new warehouse) to the right bank (old warehouse) in `leftToRight_i` minutes.
- Pick a box from the old warehouse and return to the bridge in `pickOld_i` minutes. Different workers can pick up their boxes simultaneously.
- Cross the bridge from the right bank (old warehouse) to the left bank (new warehouse) in `rightToLeft_i` minutes.
- Put the box in the new warehouse and return to the bridge in `putNew_i` minutes. Different workers can put their boxes simultaneously.

A worker `i` is **less efficient** than a worker `j` if either condition is met:

- `leftToRight_i + rightToLeft_i > leftToRight_j + rightToLeft_j`
- `leftToRight_i + rightToLeft_i == leftToRight_j + rightToLeft_j` and `i > j`

The following rules regulate the movement of the workers through the bridge :

- If a worker `x` reaches the bridge while another worker `y` is crossing the bridge, `x` waits at their side of the bridge.
- If the bridge is free, the worker waiting on the right side of the bridge gets to cross the bridge. If more than one worker is waiting on the right side, the one with **the lowest efficiency** crosses first.
- If the bridge is free and no worker is waiting on the right side, and at least one box remains at the old warehouse, the worker on the left side of the river gets to cross the bridge. If more than one worker is waiting on the left side, the one with **the lowest efficiency** crosses first.

Return *the instance of time at which the last worker* **reaches the left bank** *of the river after all n boxes have been put in the new warehouse.*

Example 1:

```
Input: n = 1, k = 3, time = [[1,1,2,1],[1,1,3,1],[1,1,4,1]]
Output: 6
Explanation:
From 0 to 1: worker 2 crosses the bridge from the left bank to the right bank.
From 1 to 2: worker 2 picks up a box from the old warehouse.
From 2 to 6: worker 2 crosses the bridge from the right bank to the left bank.
From 6 to 7: worker 2 puts a box at the new warehouse.
The whole process ends after 7 minutes. We return 6 because the problem asks for the instance of time at which the last worker
```

Example 2:

```
Input: n = 3, k = 2, time = [[1,9,1,8],[10,10,10,10]]
Output: 50
Explanation:
From 0  to 10: worker 1 crosses the bridge from the left bank to the right bank.
From 10 to 20: worker 1 picks up a box from the old warehouse.
From 10 to 11: worker 0 crosses the bridge from the left bank to the right bank.
From 11 to 20: worker 0 picks up a box from the old warehouse.
From 20 to 30: worker 1 crosses the bridge from the right bank to the left bank.
From 30 to 40: worker 1 puts a box at the new warehouse.
From 30 to 31: worker 0 crosses the bridge from the right bank to the left bank.
From 31 to 39: worker 0 puts a box at the new warehouse.
From 39 to 40: worker 0 crosses the bridge from the left bank to the right bank.
From 40 to 49: worker 0 picks up a box from the old warehouse.
From 49 to 50: worker 0 crosses the bridge from the right bank to the left bank.
From 50 to 58: worker 0 puts a box at the new warehouse.
The whole process ends after 58 minutes. We return 50 because the problem asks for the instance of time at which the last worke
```

Constraints:

User Accepted:	0
User Tried:	1
Total Accepted:	0
Total Submissions:	1
Difficulty:	Hard

- $1 \le n, k \le 10^4$
- time.length == k
- time[i].length == 4
- $1 \le \text{leftToRight}_i, \text{pickOld}_i, \text{rightToLeft}_i, \text{putNew}_i \le 1000$

JavaScript   ▾

```javascript
const findCrossingTime = (n, k, time) => {
    let collect = new MinPriorityQueue({ compare: (x, y) => x.startTime - y.startTime });
    let waiting = new MinPriorityQueue({
        compare: (x, y) => {
            if (x.endTime != y.endTime) return y.endTime - x.endTime;
            if (x.LR + x.RL != y.LR + y.RL) return (y.LR + y.RL) - (x.LR + x.RL);
            if (x.idx != y.idx) return y.idx - x.idx;
        }
    });
    for (let i = 0; i < time.length; i++) {
        let item = { startTime: 0, endTime: 0, idx: i, LR: time[i][0], pickOld: time[i][1], RL: time[i][2], putNew:
time[i][3] };
        collect.enqueue(item)
    }
    let clock = 0, np = n;
    while (1) {
        if (waiting.size() == 0 && collect.front().startTime > clock) clock = collect.front().startTime;
        while (collect.size() && collect.front().startTime <= clock) waiting.enqueue(collect.dequeue());
        let cur = waiting.dequeue();
        if (cur.endTime != 0 || np != 0) {
            let nextClock = 0;
            if (cur.endTime == 0) {
                np--;
                cur.endTime = 1;
                nextClock = clock + cur.LR;
                cur.startTime = clock + cur.LR + cur.pickOld;
            } else {
                if (--n == 0) return clock + cur.RL;
                cur.endTime = 0;
                nextClock = clock + cur.RL;
                cur.startTime = clock + cur.RL + cur.putNew;
            }
            clock = nextClock;
            collect.enqueue(cur);
        }
    }
    return res;
};
```

☐ **Custom Testcase**   (Use Example Testcases)

● Run   ☁ Submit

Submission Result: _Accepted_ (/submissions/detail/873834491/) ❓   (More Details ❯ (/submissions/detail/873834491/))

Share your acceptance!