Kyuuing Theory

Pure brute force

The key to this problem is to consider a fixed value of k. Consider the question, "Can k instructors service all students in the queue by time T?"; what we want is to find the smallest k for which the answer is yes.

Answering this yes/no question is much easier than the original problem. For example, you can just directly simulate the process as described.

- Assign an instructor to each of the first k students in the queue. Let `freed_at` be a list of length k such that `freed_at[i]` describes the time when the ith instructor is freed. Initially, set `freed_at[i] = a[i]`.
- While there is a student in the queue, we must figure out which instructor will service the student in front. Do an $\mathcal{O}(k)$ scan to find the instructor who is free the soonest (i.e. any t such that `freed_at[t]` is minimal).
- That student goes there next, so update the instructor's free time: `freed_at[t] += a[next student]`.
- Repeat this process until all students have been serviced.
- The total time it takes to service everyone is equal to the maximum value of any `freed_at` once all students have been assigned to an instructor, so the answer is yes if and only if `max(freed_at) <= T` at the end.

This simulation takes $\mathcal{O}(k(n-k))$ time.

Test each k from 1 to n until you find the first yes. This gives us a solution that runs in $\sum_{k=1}^{n} k(n-k) = \mathcal{O}(n^3)$ time, which is fast enough for the first two or three subtasks.

A faster simulation

Our simulation is slow because of the $\mathcal{O}(k)$ linear scan to find the minimal element of `freed_at`. We can address this bottleneck using a data structure!

We need a data structure that allows us to quickly insert new values, ask for its smallest value, and update its smallest value. A heap works—that's `priority_queue` in C++, or the `heapq` library in Python.

Now each simulation runs in $\mathcal{O}(n \lg k)$, and so our overall running time is $\mathcal{O}(n^2 \lg n)$, which should be fast enough for the first four subtasks.

Testing fewer values

Is there a way to avoid testing *all* values of k from 1 to n? Yes!

Return to the question, "Can k instructors service all students in the queue by time T?" If the answer is yes, then it will remain yes even if we make k larger; if the answer is no, then it will remain no even if we make k smaller.

The answer to the original problem is the smallest k for which the answer is yes, so we can binary search to find this k. Using binary search, we only need to test $\mathcal{O}(\lg n)$ different values of k.

This gives us an overall running time of $\mathcal{O}(n \lg^2 n)$. The solution presented here should still pass comfortably under the time limit for all subtasks, even in Python, since binary heaps have a low constant factor.