Minimum Average Wait Times

Chris Rudel, Meng Qiu, Katie Prescott

What’s the Problem?

- Hackerrank - Minimum Average Wait Time
- https://www.hackerrank.com/challenges/minimum-average-waiting-time/problem

Tieu owns a pizza restaurant that serves customers in order to get the average wait time to be at its minimum.

Summary

Traditional first come - first served approach for this restaurant could, with input of (0,3), (1,9), (2,6) yield an average wait time of 10.

But if Tieu was trying to get the minimum average wait time, he would serve in the order of (0,3), (2,6), (1,9) and get an average wait of 9.

The problem is to figure out the minimum average wait time when the amount of customers get much larger.

Input and Output

- We used a java scanner to get and parse the input.
- First input was the number of customers (n).
- The next n lines of input contained 2 pieces of information, the order time, and the cook time.
- For input, the order times and the time it showed up in the input did not correlate.

- The output is the minimum average wait time for this set of customers as an integer.

Test Cases

- (0,3) (1,9) (2,6)
- (0,9) (10, 4)
- (4,2) (1,8) (5,3)
- Other test cases we tried are too long to show. We tested a variety of test cases, with the number of customers ranging from 1 to 200, and the order and cook times ranging from 1 to 1,000,000,000.

Our Approach

At first we thought about having the wait time be sorted by the cook time as the example provided seemed to do just that. We quickly realized that method does not cover all cases like if people come it at times 0 and 10.

Realizations

3 classes: Main, Customer, and MinHeap

MinHeap:

swap(int fst, int snd), add(Customer c), removeTop(), isEmpty()

The Algorithm

- MinHeap add automatically.
- Sorts the heap by the cookTime.
- Adds wait time of each customer to the total wait time.
- Returns average wait time.

```
public static void main(String[] args)
{
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();
    Customer[] c = new Customer[n];
    for (int i = 0; i < c.length; i++)
    {
        int orderTime = sc.nextInt();
        int cookTime = sc.nextInt();
        c[i] = new Customer(orderTime, cookTime);
    }

    Arrays.sort(c, (c1, c2) -> c1.orderTime - c2.orderTime);

    MinHeap waitings = new MinHeap();
    long currentTime = 0;
    long totalWaitingTime = 0;
    int index = 0;
    while (!waitings.isEmpty() || index < c.length)
    {
        while (index < c.length && c[index].orderTime <= currentTime)
        {
            waitings.add(c[index]);
            index++;
        }
        if (waitings.isEmpty())
        {
            currentTime = c[index].orderTime;
            continue;
        }

        Customer served = waitings.removeTop();
        currentTime += served.cookTime;
        totalWaitingTime += currentTime - served.orderTime;
    }
    System.out.println(totalWaitingTime / c.length);

    sc.close();
```

Demo