Lucas Hasting — CS 430, SP25

Homework #1 — June 19, 2025

You are working part-time in an electronics supply warehouse, and your manager has discovered that you are studying algorithm design and wants your help in solving a problem. There is only one customer service person working at a time on the sales counter and often there are several customers waiting to order and receive parts. Your manager has asked you to devise an algorithm for serving customers that minimizes the total waiting time.

Suppose you have n customers who are ready to order parts, and you know in advance that each customer c_i will require t_i minutes. Your manager wants to minimize the total waiting time T. Where

$$T = \Sigma_{i=1}^{n}(t_i).$$

a.) Argue that the problem satisfies the criteria for using a greedy approach.

Answer. For the problem to satisfy the greedy approach, it must satisfy the Greedy Choice Property and the Optimal Substructure Property.

Greedy Choice Property: The global optimum for each person's minimized total waiting time can be found by getting the time of the previous person in line and adding it to the current person's time waiting in line (a local/greedy choice).

Optimal Substructure Property: The optimal solution to the current person is found by getting the optimal solution to the people prior (unless the person is the first person in line). Also, the main problem can be solved by solving the problem for the people before (a smaller problem).

b.) Devise a greedy algorithm to minimize the total waiting time.

Assume init_array_with_zeros(n) is a function that returns an allocated an array $A[1...n]$ where every item in A is 0. It has a worst-case time complexity of n.

Assume sort_asc(A) is a function that sorts A in ascending order using the merge sort algorithm, which has a worst-case time complexity of $n\lg(n)$.

```
//input: Array of time people will take in line
//output: Array of minimized total time people will take in line
procedure minimizeTime(A[1...n]) //n > 1
        I := init_array_with_zeros(n)
        sort_asc(A)
        time_waited := 0
        for i := 1 to n:
            timeWaited := timeWaited + A[i]
            I[i] = timeWaited

        return I
end procedure
```

c.) Provide a convincing argument that your algorithm is correct.

Answer. I will use a loop invariant to show the algorithm to be correct. I will choose the loop invariant to be the array of minimized time per person to be $I[1...i-1]$

First (initialization), when the loop is initialized, $i = 1$, so the array of minimized time per person is $I[1...0]$ which means no person has been served. This is true as the loop has not yet begun.

Next (maintenance), we will assume i is increased at the end of line 6, and show the loop invariant is true at the beginning of the loop and after a repetition of the loop. We will assume that $n > 1$, and the guard is $i \leq n$. Since $i = 1$ at the beginning of the first iteration, the array of minimized time per person is $I[1...0]$ is true, so both the guard and the invariant are true before the first iteration of the loop.

During each iteration, the time a person waits in line is $I[n-1] + A[n]$, meaning that the minimized time per person after each iteration is $I[n-1] + A[n]$, which is stored in $I[n]$. This excludes the first iteration, which stores $A[1]$ in $I[1]$. Also, the times for each person is minimal because A is sorted in ascending order before the loop begins. So, after the first iteration, the minimized time per person is $I[1...1]$. Then i is increased by 1 (it is assumed to be after line 6 within the loop), which shows the loop invariant as being true as $I[1...i-1]$ becomes $I[1...2-1] = I[1...1]$.

Third (goal), we will check to see if we reached our goal. The guard statement is false whenever $i > n$, so when $i = n$ (the last increment of i in the loop), $I[1...n]$ is the array of minimized time per person, this is true as at the end of the loop $I[n]$ is the total waiting time. Therefore, the negation of the guard and the loop invariant are both true.

Finally (termination), since i increases, it will eventually reach n as $1 \leq n$ which is the initialization of i, and i increases every iteration of the loop, so the guard will start true and eventually become false. So, when $n > 1$ the loop will terminate.

d.) Show the worst-case performance of your algorithm.

Answer. The cost table for the minimizeTime procedure is shown below:

line	cost
1	n
2	$n\lg(n)$
3	1
4	$n+1$
5	n
6	n
7	1

Taking the sum of all the costs gives:

$$T(n) = 3 + 4n + n\lg(n).$$

We then remove constants to see what happens as n gets larger, which gives us:

$$T(n) = n + n\lg(n) \in O(n\lg(n)).$$

Source: class notes