CS 477/677 Analysis of Algorithms

Homework 6

Due November 17, 2020

Note: for the programming question upload separate files in the .c or .cpp format. Include any specific compiling instructions in the top comments section of your files.

1. (U & G-required) [100 points]

A computing system at your company can analyze several megabytes of information during each day. Every day during the course of an *n*-day interval the system receives a certain amount of data: on day *i* it receives x_i megabytes. For each megabyte processed the company receives a fixed amount of revenue. Data that is not processed at the end of the day will become unavailable (i.e., it cannot be processed in future days).

Your computing system cannot process all the data it receives in one day due to its internal constraints, which limit processing to a fixed amount of megabytes per day. Furthermore, the amount of data that the system can process goes down with every day that goes by since the last day the system was rebooted. For example, the first day after a reboot the computer system can process s_1 megabytes, on the second day after reboot it can process s_2 megabytes and so on, up to s_n. We also know that $s_1 > s_2 > \ldots > s_n$. On a certain day *i* the system can process no more than x_i megabytes, irrespective of how fast the system is. To bring the system back to full performance you can choose to reboot it. However, on a day when you choose to reboot the system you cannot process any data for our company.

The problem. Given amounts of data to be processed $x_1, x_2, x_3, \ldots, x_n$ for an interval of *n* days, and given the known performance of the system $s_1, s_2, \ldots, s_n$, (considering a system that has just been rebooted on the first day), give a schedule for the days in which you would like to reboot the system in order to maximize the total amount of data processed for your company.

For example, suppose that n = 4, and the data from the following table is available:

	Day 1	**Day 2**	**Day 3**	**Day 4**
x	10	1	7	7
s	8	4	2	1

The optimal solution would be to reboot only on day 2. With this schedule, on day 1 you process 8 megabytes, on day 2 no data is processed, on day 3 you process 7 megabytes and on day 4 you process 4, for a total of 19. If you had not rebooted the system on any days, then the total amount of data processed would have been 8 + 1 + 2 + 1 = 12, which is less than the schedule above.

(a) [20 points] Write a recursive formula of an optimal solution to this problem (i.e., define the variable that you wish to optimize and explain how a solution to computing it can be obtained from solutions to sub-problems). **Submit**: the recursive formula, along with definitions and explanations on what is computed.

(b) [30 points] Write an algorithm that computes an optimal solution to this problem, based on the recurrence above. Implement your algorithm in C/C++ and run it on the above values.

Submit:

- A printed version of the algorithm (name your algorithm **schedule.c** or **schedule.cpp**).
- A printout of the table that contains the solutions to the subproblems, run on the values given above (print the entire table!)

(c) [20 points] Update the algorithm you developed at point (b) to enable the reconstruction of the optimal solution, i.e., in which days you plan to reboot the system. (Hint: use an auxiliary table like we did in the examples in class.) Include these updates in your algorithm implementation from point (b).

Submit:

- A printed version of the algorithm (name your algorithm **schedule_1.c** or **schedule_1.cpp**).
- A printout of the values that you obtain in the table containing the additional information needed to reconstruct the optimal solution, run on the values given above (print the entire table!)

(d) [30 points] Using the additional information computed at point (c), write an algorithm that outputs the days in which you choose to reboot the system (only the reboot days should be displayed). Implement this algorithm in C/C+.

Submit:

- A printed version of the algorithm (name your algorithm **schedule_2.c** or **schedule_2.cpp**).
- A printout of the **solution** to the problem, i.e., the optimal *schedule*. This should be a list of days when the system will be rebooted.

2. **(G-required)** [20 points] Write pseudocode for an algorithm that, given an interval *i*, returns an interval overlapping *i* that has the **minimum low endpoint**, or *T.nil* if no such interval exists. **Hint:** use an interval tree.

Extra Credit

3. [20 points] Whenever we reference the *size* attribute of a node in either OS-SELECT or OS-RANK, we use it only to compute a rank. Suppose that we decide to store in each node its rank in the subtree of which it is the root. Show how to maintain this information during insertion and deletion. (Remember that these operations can cause rotations).