DSA: Assignment - 1

September 8, 2022

Abstract

In this assignment you will implement multiple algorithms to sort data, as well as schedule jobs. For each task, you can download the source code that handles the input and output from Canvas or DOMjudge for `C++` or `Python3`. Since it will be used for your grading, we strongly advise you to use it in order for your program to have the correct output.

1 Tasks

You are allowed to use the available data structures of the language you chose in order to handle the list manipulation. (i.e: python `list`, python `dict`, cpp `std::vectors`, cpp `std::deque` etc...). However, you have to **implement the sorting algorithms yourself**. The use of libraries to perform the sorting itself will result in you getting **0 pt** for the given task. **You are expected to implement at least 3 different sorting algorithms to solve the tasks.**

1.1 Task 0: Warm Up (0 pt)

Operator overloading can be used to perform operations such as comparisons or arithmetic operations between objects of user defined classes *(in both C++ and Python3)*. This task does not count towards your final grade but should be relatively fast to complete.

Since throughout the rest of the assignment you will need to perform such operations between object in order to sort them. Overloading the operators can prove extremely useful to maintain readable code. For those of you already familiar with the above concept, feel free to skip this task.

Example 1. *The program takes as input: `<int>␣<comparison operator>␣<int>`. Where ␣ represents a space character. The expected output is (1, 0) for (True, False) respectively.*

Input: 1␣<␣2

Output: 1

1.2 Task 1: Supply Chain Management Software at IKEA (2 pt)

IKEA has finally decided to invest in digitising their operations and the supply chain. They contracted you to design an algorithm to sort a list of orders by selection time (`t_selection`, finding the good in the warehouse and bringing it to the surface) plus shipping time (`t_shipping`, constant). The customer orders can be retrieved (in the same order as placed) from a server database. You should expect between 100-10K elements.

Example 2. *The program takes as input a data-set of orders where* `id, t_selection` *and* `t_shipping` *are of type* `unsigned int`, `n` *is the number of orders, and ␣ a space character.*

<id_1>,␣<$t\_selection_1$>,␣<$t\_shipping_1$>;␣...;␣<id_n>,␣<$t\_selection_n$>,␣<$t\_shipping_n$>`\n`

The expected output is a space separated list of the `ids`, *sorted by* `t_selection + t_shipping` *and terminated by a newline* `\n`.

Input: 1,␣500,␣100;␣2,␣700,␣100;␣3,␣100,␣100`\n`

Output: 3␣1␣2`\n`

1.3 Task 2: Multiple Warehouses (2 pt)

Your sorting program is successful, IKEA asks for your solution to run for several IKEA warehouses. You should expect between 10*10K - 20*20K elements.

The program has the same input/output data format as Example 2.

1.4 Task 3: Include Past Data with one Warehouse (2 pt)

Your proposal takes off, but IKEA decides to use your software in all the warehouses in The Netherlands to also simulate past customer orders (which are not stored in the order as placed, but randomly) and optimise future delivery. You should expect between 10*10K - 20*20K elements.

The program has the same input/output data format as Example 2.

1.5 Task 4: Include Past Data with Multiple Warehouses (2 pt)

IKEA decides to sort the past orders from all warehouses in the Netherlands also based on the Warehouse location (e.g., Amsterdam, Utrecht, Delft,...). You should expect between 10*10K - 20*20K elements.

The program has the same input/output data format as Example 2.

1.6 Task 5: Job Priority Scheduling Sort (2 pt)

Covid happens and IKEA is in turmoil, as the way they were calculating the job priority (i.e., the constant selection time above) does not work anymore. You have a bunch of jobs to schedule on a single 'machine'. Job j requires p_j units of processing time and has a positive weight w_j which represents its relative importance - think of it as the inventory cost of storing the raw materials for job j for 1 unit of time. If job j finishes being processed at time t, then it costs `t *` w_j dollars. The goal is to sequence the jobs so as to minimize the sum of the weighted completion times of each job. You should expect between 10*10K - 20*20K elements.

Use **Smith's rule**, that is, schedule the jobs in the order of their ratio of processing time to weight. This greedy rule turns out to be optimal.

Example 3. *The program takes as input a data-set of jobs where* `id, p, w` *are of type* `unsigned int`*,* `n` *is the number of orders, and ␣ a space character.*

<*id*$_1$>,␣<*p*$_1$>,␣<*w*$_1$>;␣...;␣<*id*$_n$>,␣<*p*$_n$>,␣<*w*$_n$>\n

The expected output is a space separated list of the `ids`*, sorted using* ***Smith's rule*** *and terminated by a newline* **\n***.*

1.7 Bonus: Dynamic Job Priority Scheduling Sort (1 pt)

Your scheduler can handle new orders coming in at any time of day. For this task you are allowed to use `PriorityQueue` in Python and `priority_queue` in CPP to handle the dynamic sorting (see boilerplate).

Example 4. *The program has similar input/output data format to Example 3. The difference is that you will receive multiple data lines to add to the scheduling priority queue:*

Input: 1,␣500,␣100;␣2,␣700,␣100;␣3,␣100,␣100\n

Output: 3␣1␣2\n

Input: 4,␣200,␣100;␣5,␣400,␣100;␣6,␣600,␣100\n

Output: 3␣4␣5␣1␣6␣2\n

⋮

2 DOMjudge

There will be **separate contests** for C++ and Python3. Pay special attention to submit your programs for testing on the correct one as timing constraints differ per language.

For this assignment half test cases will be and downloadable from the user interface, and half will be secret.

Read the DOMjudge team manual for further information.

3 Deliverable

You only need to implement one of the two languages for each exercise, `C++` or `Python3`.
The expected deliverable for this assignment is a **<student-number>.zip** archive containing the source code files of all the tasks you have implemented. All the tasks of the assignment need to be implemented using the same language (eg you can not implement task 1 in Python and task 2 in C++).

You need to upload the archive on Canvas. Submissions on Domjudge are purely for testing purposes and **do not** count as official submissions. **Failing to submit on canvas will result in you failing the assignment.**

4 Hints

1. Consider the format of the data for each task. Ask yourself the following questions by looking at the sample tests.
   - Is the data ordered? partially ordered? or unordered?
   - Does the task require the algorithm to be stable?
2. For Task 5 you can use one of your previously implemented algorithms and adapt it to follow `Smith's rule` when sorting.
3. Perform floating point operations/comparisons to find the proper ordering.
4. Use what you learned in the Warm Up task to improve your code's quality.