**Final Project Report**

**The Issue**

A common problem among individuals in this day and age is the immense amount of tasks we consistently have on our plates. I can’t speak for others, but throughout high school and now college, I have been repeatedly told that the key to adapting to this and even thriving is that **“you need to prioritize.”** Yet, many people, even within my own family and circle of friends, have trouble knowing just what to prioritize or even how to do it in the first place. Thus, we have…

**The Solution**

My take at a solution to this problem that plagues many individuals is what I’ve started calling the **Prioritizer**. Its premise is simple: the user will input whichever tasks they are currently being faced with, along with each task’s deadline (a date in MM/DD format), time it takes to complete (in minutes), and urgency (marked as 1, 2, or 3 according to the below scale). The scope of this program, or what it’s intended to work best with, is a weekly or even monthly list of tasks where deadlines are limited to a date.

**Urgency Legend:**

* 1: Not Important: The task should still be completed if possible, but it’s given the lowest priority.
* 2: Important: The task should be completed. This is the “average” priority.
* 3: Critical: The task MUST be completed by the deadline, so it’s given the absolute highest priority.

From the technical side of things, the overall algorithm was actually quite complex and took numerous hours to come up with. This was likely due to there probably being an astounding number of ways to do this and nearly infinitely many routes to take and variables to include if you let yourself go down that rabbit hole, which as you can probably tell, I did at first.

In the end, instead of trying to conjure up some complex solution for every variable imaginable, I sat down, analyzed how I normally prioritize tasks myself (I’m a big planner by nature), and came up with the following:

**The Prioritizing Algorithm Concept**

Deadline > Time to Complete > Urgency

This is the order that the variables will be used to sort/order the tasks by. Now, to be efficient and even work in the first place, the specifics involved a decent number of the concepts of this course (Algorithms).

1. The tasks had to **first be sorted by their deadlines** to break down the task list into chunks so that the tasks could even be further prioritized by time to complete and urgency in the first place. Thus, this step involved a double dose of **Transform & Conquer**.
2. The next step involved **traversing the Task List and, as each deadline block was passed, passing them to be quick sorted, first by time to complete and second by urgency**. The sorting for each step had to be efficient in the event that there were a large number of tasks and/or that the algorithm ever needed to scale up to a group or even organizational level. For this, I figured quick sort would be best as it’s an in-place algorithm and supposedly works better than merge sort with array-type data structures. Thus, the concept of **Divide & Conquer**.

The last few general Algorithms concepts applicable to and utilized in this project include:

* **Efficiency** (ideal and even important in any computational sense; always a good thing to strive for and so I did with this project)
* **Data Structures** (have to manipulate arrays (read: vectors), structs, classes, etc.)
* **Brute Force** (potentially -> based on the linear traversal of the *taskList* vector to break it down into deadline blocks to divide and conquer with)

**Notes on the Variables**

The below variables are what I consider the absolute essentials to prioritizing tasks. After running through how I normally prioritize tasks myself, these are the major three that I take into account.

**Deadline:** a string with the MM/DD format; a string is much easier to work with than a date/time object in most instances, and since it’s already in a numeric and consistent format, sorting just works. Additionally, for the sake of a computational solution that wouldn’t require five years of work from a research team, having deadlines be a date as opposed to introducing exact times allows for the above computational steps to actually exist and function.

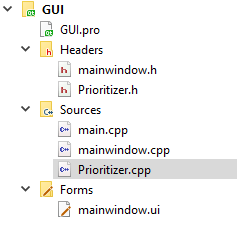
**Time to Complete:** an int whose units are minutes; by default, tasks that are shorter/take less time to complete get priority over those that take longer to complete. I believe this is probably more of a personal preference so there may be a configurable option to flip that, but that’s just generally how I do things and I find it works out well for me for reasons a computer can’t easily understand (stress management, etc. -> refer back to my trying to take into account every variable in existence).

**Urgency:** an int value of 1, 2, or 3 to help further prioritize tasks based on importance

Lastly, I figured this sort of program would provide a much better user experience if it had a GUI, so I made just that using Qt.

**Source Code**

**File Structure**

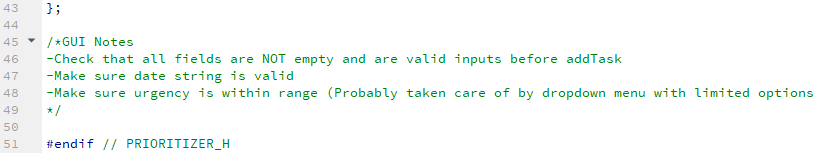


\*Prioritizer.h/.cpp are the actual algorithm/computational files; mainwindow.h/.cpp/.ui are the UI related files that had to be programmed and connected to the Prioritizer class (main.cpp is auto generated just to setup all the Qt UI stuff so I figured there’s no point to including it here).

**Prioritizer Code**

Prioritizer.h



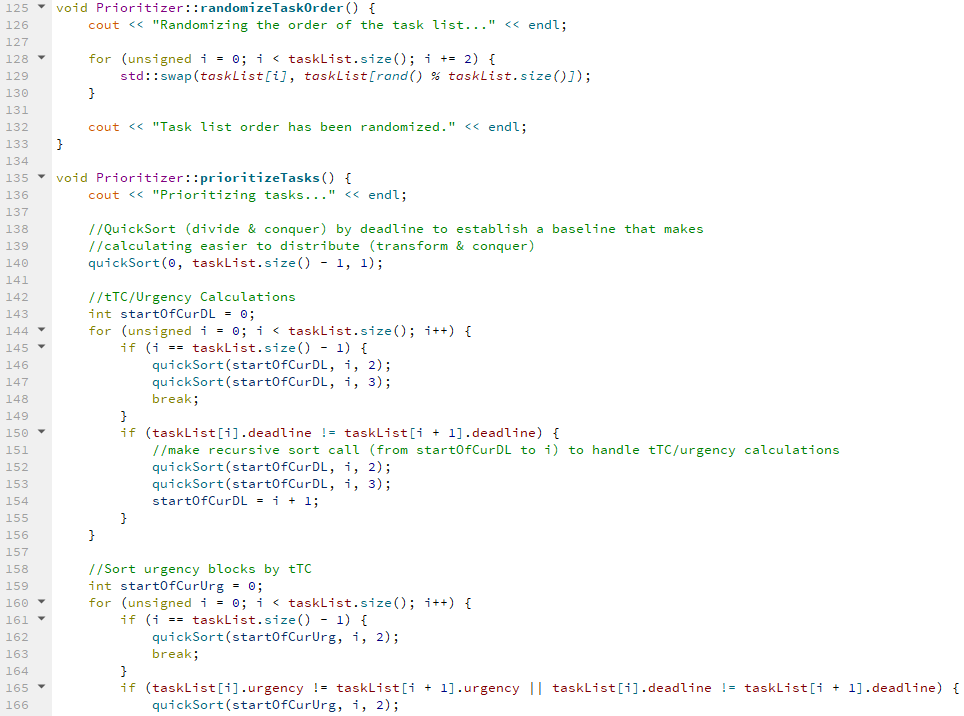


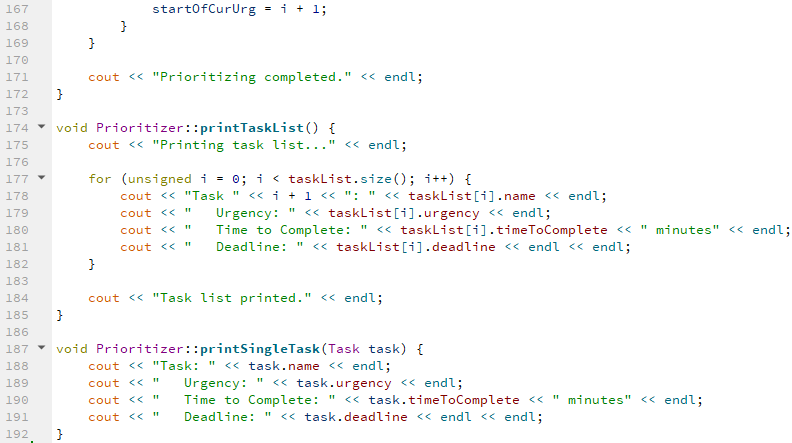
Prioritizer.cpp











**GUI (Qt) Code**

mainwindow.h



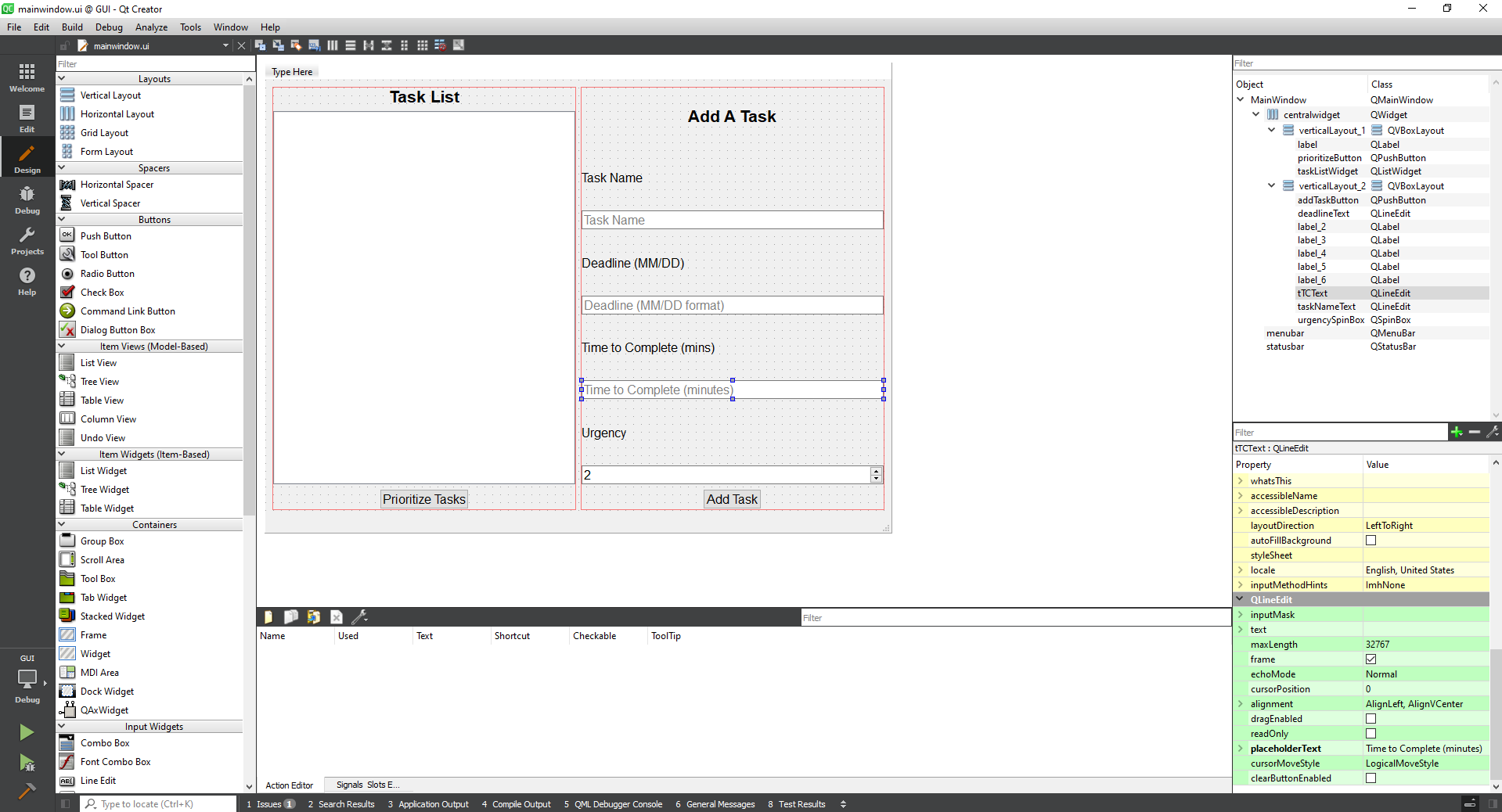
mainwindow.cpp





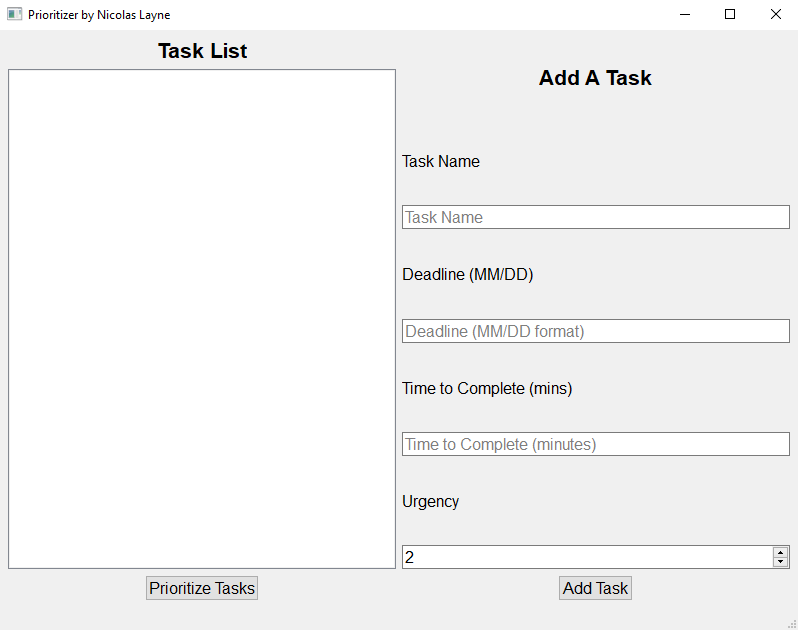


mainwindow.ui and Editor overview

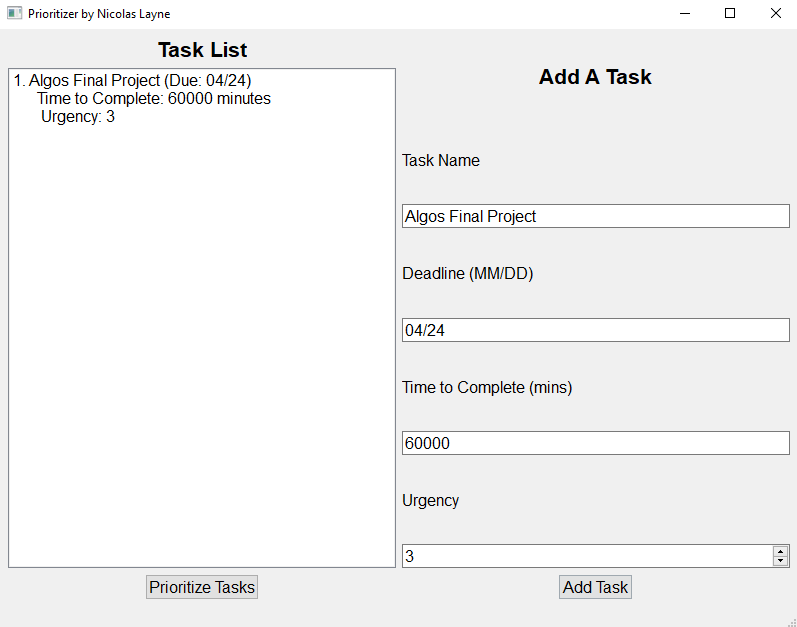


**Conclusion/Results**

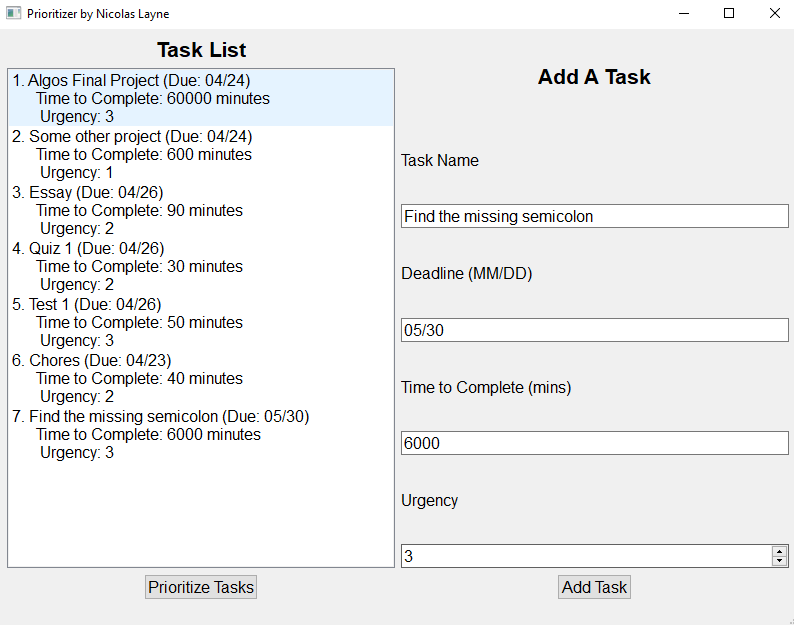
**Final Program**



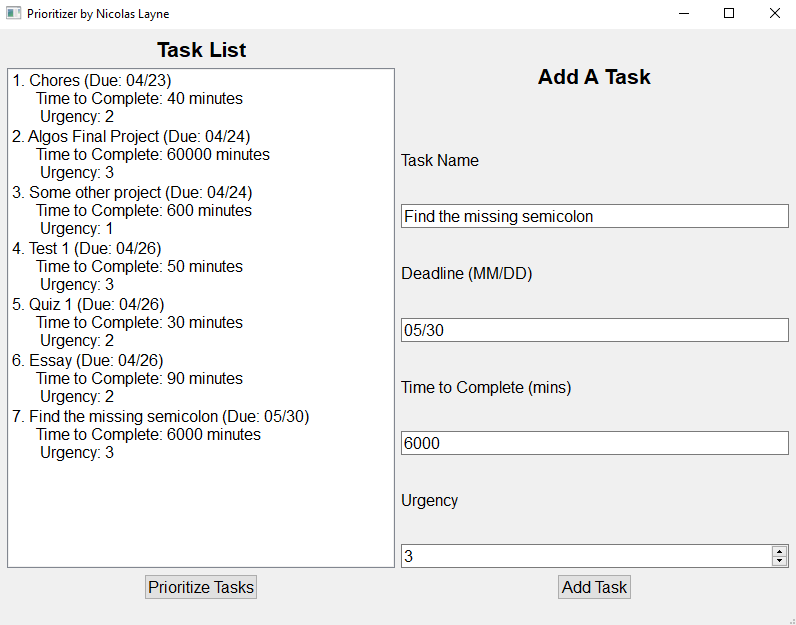
Just to reiterate, the idea of this program was that an individual who has numerous tasks to complete inputs them into this program (in any order) and then gets a list of prioritized tasks back in order to have an efficient plan to complete the tasks they put in. Here’s what that would look like:



\*Tasks are added one at a time using the right side of the UI. Each time any changes to the *taskList* are made, the UI is updated to the reflect that (all demonstrated in the link to the video).



\*Once all tasks are added, as shown above, the user will click “Prioritize Tasks” and the resulting output would be:



While I opted not to work in a pair or group, this was more due to my already tight schedule than my inability to work with others (we weren’t kidding, the projects never seemed to stop coming in!). Regardless of that, hopefully I pulled it together with what I’d like to think is a pretty cool idea and maybe even a little practical or helpful to some. I ended up putting a good number of hours into making this project, and plenty of time drawing up the concept and designing everything. I will say that having a team member or two would’ve probably helped greatly in the initial planning/design phase just to have an extra perspective or two.

All in all, I think I made something pretty cool, all while applying a number of concepts from this course, and I can only hope you agree to some extent. This concept could probably even be greatly expanded and scaled up in many different regards such as support for multiple individuals (organizational level) and even more variables in the actual prioritization algorithm. Finally, it was most certainly a pleasure to build the Prioritizer in the end, and I believe I actually learned a good deal doing it, despite the numerous moments of near insanity.