1. **Names and email ids.** You must include your name and UM email id (e.g., "weimerw"). If you have a partner, you must include the partner name and email id as well.

Nicholas Recker (nrecker)

2. **Selected project.** A brief description of the open source system to which you contributed (1 paragraph). You may reuse text from Part A.

I contributed to Notepad++, a free (in both senses of the word) lightweight code editor for Windows that feels a lot like Notepad. Its website is https://notepad-plus-plus.org/.

3. **Project context.** An analysis of the open-source project's context and "business model". This may include a short history of the project, competing open- and closed-source projects, or a discussion of the developers' motivations to build this system. Essentially, we want to know why this project exists and why it is important. (At most one half page.)

Notepad++ is completely free; even the download is freely offer from the website. It uses pure Win32 API and STL; this gives it exceptionally high speed and small size in a Win32 context. In their own words, "By optimizing as many routines as possible without losing user friendliness Notepad++ is trying to reduce the world carbon dioxide emissions. When using less CPU power, the PC can throttle down and reduce power consumption, resulting in a greener environment." In summary, Notepad++ is a very lightweight code editor for Windows.

4. **Project governance.** Describe the processes and tools the project uses to communicate coordinate among contributors. Are these processes formal or informal? Provide an explicit description (possibly with a diagram) of the acceptance process used for efforts like the task you completed. If applicable, include standards or expectations regarding software engineering activities including requirements, design, and quality assurance. Alternatively, mention that no such standards exist. (Usually about one page; length varies.)

Besides the standard github interface, Notepad++ uses a separate forum. However, there seems to be little native development discussion there. Rather, the forum focusses on news, assisting users, and plugin development. (In retrospect, perhaps a plugin would have been a better vector for my changes).

Notepad++ seems to be rather informal. The vast majority of accepted pull requests are for language (as in German, French, etc.) updates. Of these, most are simply requested, then reviewed and accepted by the project owner. Often the review is silent; from a third party's perspective the change is simply accepted. However, other times the project owner will use Github's review feature. It is merely my conjecture that he in fact privately reviews all pull requests, but only makes it public if there are changes to be made. In the words of TimPianoDude on the forum, "the resources link on the Notepad++ web site (https://notepad-plus-plus.org/resources.html) seems to indicate that there are code reviews that occur for updates, but I can find no more than this passing reference. How is the code reviewed prior to release and by whom?"

There are some standards, detailed in the contributing.md file. These are mostly style standards though; camel over underscores, c-style braces over java-style braces, etc. Nothing of quality assurance is mentioned.

5. **Task description (per task).** A description of the tasks you have implemented and a high-level description of how you implemented them. (At most one half page per task.)

Notepad++ has a feature called “Sort as Integers”. Currently it sorts lines by the first integer on each of the selected lines breaking ties by original position. Another user has made an issue requesting that it instead perform a “Natural Sort Order”, which means alphabetical order except that multi-digit numbers are ordered as a single character.

I implemented this by writing a comparator for use by the stl sort. This comparator iteratively checks whether the first remaining character of the two strings to be compared are numerals. Then if they are both numerals, I compare the results of std::stoll. If instead they are both strings, I take the their substrings up to the first numeral and compare those. Finally, if one is a numeral and the other is a string, I compare the characters (which cannot possibly match). In either of the former two cases, an exact match is possible, so I update an index counter to the next uncompared character position (which is the same for both strings).

6. **Submitted artifacts (per task).** Evidence of the code, documentation, test cases, and/or other artifacts you produced for the task, and evidence that you submitted them to the project. We require links to publicly available resources (project repository, email archives, pull requests, etc.). Each such link should be accompanied by a brief description.

Pull request at https://github.com/notepad-plus-plus/notepad-plus-plus/pull/4413. While I didn’t submit any test cases, I did create and use some. I have uploaded my manual test cases, manual oracles, and large scale test case generating script to this Google Drive folder https://bit.ly/2ESUZBF

7. **QA strategy.** Describe which QA activities you performed and justify why you selected these QA activities over others. Describe metrics if appropriate. The justification will likely refer to relevant requirements as well as to the project's practices. (At most one page.)

Testing was a little tricky for this task. I would have liked to automate testing, but unfortunately I was not able to find any way to activate Notepad++’s sort actions without manually clicking on them. This limited the volume of testing I was able to perform. I created four small test cases with oracles before I began coding. Each was under ten lines long, and either demonstrated basic behavior (such as sorting “foo 2” before “foo 10”) or targeted a specific problem I feared (such as improperly sorting Unicode characters not found in ASCII). In practice I found that there were just as many errors in my test cases as in my actual code. However, these bugs were substantially easier to remove, perhaps because each was localized to ten “lines of code”.

I also wrote a python script to generate large, random test inputs. I chose this for two reasons. First, I had had great success in the past using random input to catch runtime exceptions, especially segfaults, in a prior project for another class. Second, Notepad++ prioritizes speed, so I wanted to make sure my changes would not slow the sort too much. I found this approach caught fewer bugs here than in my prior project; I hope this means I'm improving as a programmer rather than applying the wrong tool to the job.

I chose to rely on such ad hoc testing since the change had well-defined areas of influence; I am confident I didn't break bracket matching, even without a test for it. Given the timeframe of 24hours, I didn't want to try anything fancy like mutation testing or static analysis. This decision is further supported by the short length of the edits; I can manually test the whole thing in less time than it takes to learn how to apply fancier debugging software to a feature activated only by mouse. The project didn't seem to have any standard testing practices, so I was free to do as I saw fit.

8. **QA evidence.** Evidence of your quality assurance activities. This might include source code, links to source code of tests, test results, comments from code reviews, reports from static or dynamic analysis tools, links to or screenshots from a continuous integration platform, and so forth. For example, you might use Travis CI, which supports continuous integration testing for GitHub, and link to a CI build in your report (see examples below; while neither required nor sufficient, Travis CI evidence is particularly easy for the course staff to evaluate).

Due to the manual nature of my tests, it is difficult to show screenshot evidence of success. Instead I have screencaptured my performing the test manually and uploaded it to https://bit.ly/2ESUZBF, the same folder where I uploaded the manual test files and test input generating script. The time taken is about twice that of the old integer sort. I did not judge exact measurement useful, as it is unlikely that users will be using Notepad++ to sort hundreds of thousands of lines; data on that scale would be best handled by a spreadsheet program.

9. **Plan updates.** A description and justification of deviations between your initial plans and your performed activities, if any (there are almost always a few). Changes are expected, but they should be tracked and explained. Describe changes in scope (e.g., fewer tasks) and in the schedule and work allocation. Provide an after-the-fact "what actually happened" schedule and note differences. Explain the causes of the changes, such as unanticipated risks. (At most one page.)

I did not end up having time to implement natural sorting for decimal numbers. As it is, I didn't get around to submitting my pull request until Saturday, three days after schedule. C'est la vie. However, this was primarily because I didn't work on it early enough, not because things took longer than expected. Reading the code and finding the relevant sections of code proved inextricable. Just as I thought I had found the relevant code and moved on to reading it, I would find that it in fact offloaded the relevant logic to yet another file, and the search continued. After all was said and done, finding and reading code took twice as long as expected. In retrospect, I

should have anticipated this from my prior experiences with large projects. Secondly, building the project took much longer than expected. Beyond this however, everything took about as long as expected.

6hr	Build project
1hr	Find relevant code
3hr	Read relevant code
30min	Write tests
30min	Write test scripts
30min	Decide on algorithm/library
30min	Read contributing.md
3:30hr	Write first draft
30min	Compile
2:30hr	Debug
30min	Proofread and submit pull request

10. **Your experiences and recommendations.** Summarize your experiences (and what you learned!) interacting with this community of open source developers, focusing on any surprising or unusual aspects of the process or interaction. Did you run into any trouble understanding, changing, or contributing to a large, pre-existing project? Were there unanticipated challenges in either implementing your change or in getting the change submitted to, and accepted by, the project maintainers? Did the project collaboration process or culture help or hinder your effort in any way? Characterize any interaction you had with the team leadership and community. Highlight any useful (or useless) input you received. In addition, describe any changes you would make if you were starting a new project from scratch. What worked here (and why), what did not work here (and why), and what would you do instead? You may (but are not required to) also relate the experience from this homework assignment with relevant experience from internships or other projects. (Two or three pages. This is the heart of your report. Convince us that you integrated course concepts and learned something.)

I somewhat regret choosing Notepad++ as my project. I didn't catch this at first, but almost all the accepted pull requests in recent months are language updates, not features. Had I looked more closely, I might have chosen a different project. Oh well.

The owner of Notepad++ is definitely somewhat active (his most recent comment was 3 hours ago as I am writing this) but has not responded to my two day old pull request. Additionally, my request weeks ago to be assigned to the issue was never responded to. This suggests to me that he does not have a lot of time for this project and/or is stretched thin across the many responsibilities thereof. Thankfully another contributor had commented on the original issue explaining what was needed; had I needed to elicit requirements, things would have taken much longer. Additionally, the community has already discussed many topics on the notepad++ forum (distinct from github). This came in handy when I was struggling to build the project.

While this lack of activity has definitely hindered acceptance of my pull request, it has not affected my development. The project was well commented, so I never had need to ask for help interpreting or sorting though it. It also helped that the issue I addressed had clear keywords (sort, ascending, etc.) to search for. The beginning of finding the relevant code was just grepping for those keywords. I tried using Visual Studio's ctrl-shift-f to search the project, but many files, including the ones I was looking for, were not included in this search. I still don't know why.

This was my first real experience with Visual Studio, though I had used other IDEs like Eclipse and Netbeans in high school. I chose to use Visual Studio since all the documentation regarding how to build Notepad++ was for Visual Studio. Last summer I worked on a similarly large (albeit not open source) project that used a makefile to build. Of the two, I greatly prefer makefiles. Most of the 6 hours I spent building Notepad++ for the first time were spent messing with Visual Studio. The build instructions are for VS2013, but I was only able to find VS2015 and VS2017 from Microsoft. Thus I ended up following community instructions to modify the build files to support VS2017. The community was extremely helpful for this; everything I needed to know had already been said, so no active participation was required on my part.

If I were to start a new project from scratch, I would keep things mostly the same. I've complained a lot in the above paragraphs, but most of that is my own fault for choosing a project in its later stages of life. However, there are still a couple of things I would change.

I would support a makefile means of building the project, possibly to the exclusion of supporting IDEs. With a makefile to succinctly build the project from the command line, everything else I used Visual Studio for could be done by any other code editor, such as Vim, Emacs, or even Notepad++. This would allow developers to skip the slow startup time of Visual Studio. Keeping in mind that system latency is linked to developer actions per minute, this is a worthy goal. While not relevant for Notepad++, which is exclusive to Windows for sake of optimization, this approach would also allow greater portability, especially to Linux.

The code was very well commented and organized; I had no troubles understanding it. This leads me to believe the code review process already in place at Notepad++ is effective. However, this may not be sufficient; a good code review process must also be efficient. The project owner takes days to respond to pull requests. While this may be acceptable at later stages of the project life cycle, it is still not ideal. Were I to start such a project, I would want to invest time in an automated regression test, perhaps using Jarvis. Taken over the life cycle of a long project like this, it would save quite a bit of the owner's time reviewing faulty pull requests. As is, there is only a compilation check. Additionally, it would be nice to implement a style checker; one of the

requirements for pull requests at Notepad++ is that each PR consist of exactly one commit. This is great; single commits are easier to understand than spaghetti branches. However, I saw some pull requests use multiple commits anyway, only to be told to fix it days afterward. What an unfortunate loss of time! This is the sort of problem a computer ought to be able to catch automatically, just like a regression test.

I think I made a wise decision in choosing a small, well defined part of the project to work on. Reading the relevant code took long enough as it is; I can't imagine trying to comprehend enough of an existing large project to, say, fix a fatal, ill-described bug within a 24hour timeframe.

Random test generation worked out well here. I did have to modify my test generation script a bit after seeing what actually happened. In particular, after I scaled the input to take a reasonable amount of time to sort I found that few strings matches in their first chunk, much less their first two chunks. To remedy this, I restricted the random strings to the character set "abc" and shortened the chunk lengths. I think random test generation was absolutely necessary for proper stress testing; manually generated test cases are unlikely to match typical use cases and thus typical use times. Testing for performance was important given Notepad++'s stated goal of higher execution speed.

11. **Advice for future students.** Give a single sentence of advice to students taking this class in future semesters. This advice can either be about HW6 or any other course aspect. What do you wish you had known earlier? Any non-empty answer counts. We will display such advice *anonymously* on the webpage for next semester. If you are working with a partner, each of you should give a separate single sentence. If you are willing to let future students see your materials (cf. the examples at the bottom of this webpage), indicate as much in your report.

Just because a project is accepting pull requests doesn't mean it is accepting the kind of changes you want to make.

I am willing for future students to see my materials.