Ian Anderson

1641 E 115th St

Cleveland, OH 44106

Professor Michael Parker

Department of English

Case Western Reserve University

Guilford House

11112 Bellflower Road

Cleveland, OH 44106

December 6, 2013

Dear Professor Parker,

Please find enclosed my proposal of an Independent Study program, titled “An Exploration of Sorting Algorithms.” This document shall become the basis of a course of study, which shall aid in my studies as an undergraduate Computer Science major, and in my ability to both obtain employment and function in a work environment. Additionally, this proposal fulfills an assignment for another course, which is required for the aforementioned major.

This proposal is written for two primary audiences: you and me. This means that both technical and non-technical terminology may be used, depending on the section. Attempted in this document is the buildup of such terminology, so that one who is a non-technical reader can, by the end of the document, understand at least the basic terminology associated with this field.

There are a few absent sections in this proposal because their inclusion would have simply been a waste of space and of the reader’s time. The “Audience Involvement” section has been omitted because the only anticipated involvement I see would be that of myself and that of a Computer Science professor who agrees to mentor me. As I am the only audience member in that pair, I believe it unnecessary to include such a section in this proposal. The “Budget” section has also been omitted, and this is due to the fact that the proposed Independent Study would require no budget. It is proposed to be during a standard semester for which tuition would be paid regardless, and the proposed coursework would require only time on existing computers owned by the university and already available to students, something that costs no money.

Thank you for the time spent reading this document and considering its contents. For any questions, please contact me by email (isa12@case.edu).

Sincerely,

Ian Anderson

ENGL 398 Proposal:

An Exploration of Sorting Algorithms

Independent Study

Ian Anderson

December 4, 2013

Submitted to:

Professor Michael Parker

*Abstract*

This study is intended to further the author’s studies in Computer Science, particularly in the realm of algorithms. The study will be useful both during and after school, as he is currently majoring in Computer Science and intends to pursue a career in software development, where thorough knowledge of such algorithms should prove valuable. The proposal contains details of what the project will be, the literature involved, and the intended schedule of research for this study. Over the course of fifteen weeks studies will be conducted with the goal of developing a novel sort, hopefully one that is superior in at least one situation to others.

Table of Contents

Project Description………………………………………………………………………………4

Literature Review………………………………………………………………………………..4

Research Plan……………………………………………………………………………………6

Qualifications……………………………………………………………………………………7

References……………………………………………………………………………………….9

1. *Project Description*

I am proposing research on various sorting algorithms, in particular the attributes of those algorithms that are efficient, either in space or in time. This topic is of some importance to me, due to a strong interest in computing and the coursework I am currently taking. This coursework includes both EECS 340 – Algorithms and Data Structures, and EECS 293 – Software Craftsmanship.

To work effectively in this research, a powerful computer or group of computers would be necessary for benchmarking the speeds of these sorts. My request to use a powerful computer is due to the need to stress test each algorithm on large data sets, some of which would be generated during the test. I expect this need to be easily satisfied by designating time to work in the Alienware lab in the Olin building – a more powerful computer than these should not be necessary.

Though the task would be quite difficult, the goal of this project will be to create a better sort. The greatest potential resulting from this research may be in creating a quicker in-place sort – one which does not use auxiliary memory in which to do the sorting – as these sorts are slow by comparison to those that do use this extra space, but are advantageous in situations where memory is limited.

I expect only minimal advising would be necessary. Such needs as securing lab resources and choosing of particular algorithms as examples of their types of sort are the primary reasons I would have need of an advisor.

The groundwork for this project would be implementation of various sorting algorithms on the test machine, followed by generation of large data sets. These data sets shall include pre-sorted data, nearly-sorted data, data that is sorted in reverse order, data with few unique values, and random data. Once this is complete, a resource monitoring and logging application would be needed to measure the behavior of these algorithms as they process the data. Finally, a large amount of time on the test machine would need to be allocated for running the tests.

When these tasks are completed, picking out the smallest of the algorithms in terms of time and memory consumption would begin the process of examining the best sorts. Afterward, careful analysis of the specific behavior of each algorithm would be required to determine similarities between them. The project would end with some considerable time spent attempting to combine these features into, hopefully, an even better algorithm.

1. *Literature Review*

Many algorithms for other procedures, such as efficient searching of a data set, depend on having sorted data [1]. Since a programmer’s choices in that situation are to either hope the data given is sorted or sort it before using it, quick sorting algorithms are very important. Additionally, many systems use small, embedded chips which have very small amounts of memory and cannot afford algorithms that need more. An example of one of these systems is the microphone and speaker system in a space suit’s helmet.

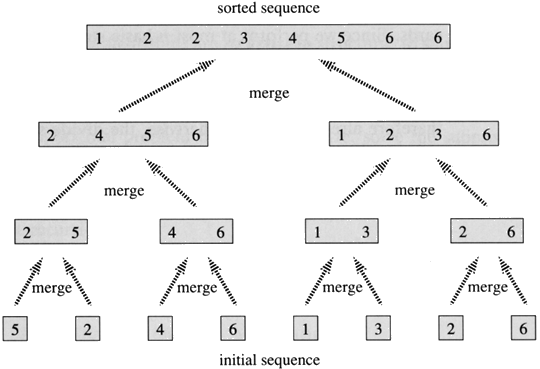
Sorting algorithms have been largely the same for quite some time now, with the generally accepted “optimal” general-purpose sort being Quick Sort [2, 3]. This algorithm, however, is outdone in most cases by others. The difficulty is in knowing the data well enough to be sure of which algorithm is best. Quick Sort has had several improvements made to it since its debut by Tony Hoare [2, 3, 9]. Unfortunately, in cases where memory is significantly limited, Quick Sort is a particularly poor choice as its recursion makes it need log2 of the input size in extra memory, just to get through all the function calls. Another algorithm for sorting in little time, as Quick Sort is designed to do, is called Merge Sort. This sort, though, is also recursive and thus uses relatively large amounts of memory. Specifically, Merge Sort uses more space – it is directly proportional to the input rather than a logarithm of the input – but is a bit faster in some cases.

Figure 1: The “merge” procedure of Merge Sort. After being broken down into single elements, the list is merged together by adding the smallest element from the two sublists being sorted to the merged sublist until the entire list has been merged back together [10].

This problem is resolved by using an in-place, iterative sort, of which Insertion Sort is a well-known example [6, 8]. This type of sort does not have to use memory for calls to itself, and uses a maximum amount of auxiliary memory equal to the size of an element from the list it sorts. However, these sorts are significantly slower for large input than the recursive approaches. A few sorts, such as Tim Sort, have been developed that hybridize the two, using for example Insertion Sort on small subsets of data, and Merge Sort on the results of these Insertion Sorts [5]. This approach is better on runtime for large data than Insertion Sort, and better on storage than a standard Merge or Quick Sort.

Another approach that can be taken is called Counting Sort. This sort is significantly faster than the previously mentioned sorts, assuming that certain things are known about the data [4]. First, it must be sorting integers only. Second, the largest integer being sorted must not be significantly larger than the number of integers being sorted. When these are the case, the time taken to sort the list is directly proportional to the size of the input, but regardless of those conditions the auxiliary space taken is directly proportional to the largest number being sorted. So, this method, too, fails at being both fast and memory-efficient. Radix Sort avoids some of the time issue caused by large numbers by first placing them into “buckets” based on, for example, their least significant digit, but still has similar limitations [4].

Getting closer to a good solution, programmers can use the Schwartzian Transform, or mapping of complex data to simple data and sorting the complex data based on its corresponding simple version, in order to take advantage of things like Counting and Radix Sorts for non-numerical data [7]. This generally improves both the time and memory necessary for the sort, because there are far fewer comparisons to be made on the simple data and this data usually takes up considerably less space than the complex data does, so its cost in that regard is negligible in most cases.

After this survey of the field, I believe it worthwhile to delve further into study of each individual algorithm that is in use today, and to study what unifies those algorithms that are considered “good” for their various applications.

1. *Research Plan*

This project is intended to take the span of one semester, be it Spring or Fall of 2014. This means that there will be approximately fifteen weeks during which to accomplish the goals of the Independent Study. These goals include achieving familiarity with a wide variety of sorting algorithms and developing a novel sort, which would have a less emphasized goal of being better than those that already exist in some way.

In the first five weeks, the study will be focused around further research into the topic of sorting algorithms. This will be primarily accomplished by self-assigned readings, including those citations from this proposal that are text. The purpose of these readings will be to further familiarize myself with the subject matter, as well as with the type of thinking involved in creation of an algorithm. For example, Vladimir Yaroslavskiy’s email correspondence with Joshua Bloch and Jon Bentley will be used due to its introduction of a novel implementation of quicksort and its thorough mathematical explanation of why this implantation is better than the classical one [3]. This email also explains many different ways to test an algorithm, which will be useful during a later portion of the study.

In addition to the readings, the first five weeks shall include practice of implementation of sorts. Specifically, each of these weeks I will read the description of two algorithms, attempt implementation, read the algorithms’ pseudocode – a translation of the real code into something between code and English that is used to explain specifics of implementation without programming language dependency – and make another attempt if the initial one is inconsistent with the new information.

The next five weeks will begin with ensuring that the implementations I have created are efficient. If unsure, this is one of the times where Professor Koyuturk’s mentoring would be greaty appreciated. After this has been completed, I will develop a test suite for the algorithms I have implemented. The list of types of tests will be that described in Vladimir Yaroslavskiy’s email: “{ 100, 1000, 10000, 1000000 } x { sawtooth, rand, stagger, plateau, shuffle } x { ident, reverse, reverse\_front, reverse\_back, sort, dither}” [3]. The first set of brackets contains the sizes of data sets, the next, the types of data, the last, modifications made to the data after generation. Specifics on what these types and modifications are, exactly, are included in the email directly below the quoted section.

The final portion of this second set of five weeks will involve little work. Instead, they will be comprised of trips to the Olin Alienware lab and running the tests described above on one of those computers for the various algorithms being investigated.

The next week will be spent compiling the data gathered in the previous weeks into a spreadsheet to look at what, specifically, the different algorithms are good for. This will be used in the last four weeks as a point of comparison for the sort that I will attempt to develop.

The three weeks after that will be dedicated to the new sort’s development. During these weeks I shall draw on the knowledge acquired through this Independent Study to endeavor to combine the best features of the various sorts into one. This lofty goal will likely not be attained, but at the very least some new sort will be created, and it may be that this algorithm is superior in some way to the others.

The last week of the semester will be spent reviewing the algorithm I will have created and running it through the same test suite used for the other algorithms. The Independent Study will terminate with a comparison of my algorithm to the others to determine what it is best at and which algorithms I have beaten.

1. *Qualifications*

My current standing is as a junior Computer Science major. This means that I have a familiarity with the concepts of programming and that I am delving into the finer aspects this year. Particularly relevant courses to this study that I have participated in are EECS 233 and EECS 340, the former completed and the latter in progress, which both deal with algorithms. EECS 233 is largely about the practice of programming, and the data structures and algorithms involved. EECS 340 is a primarily theoretical course dealing almost exclusively with algorithms.

Additionally, I have a strong interest in this area. I have had already one summer position as a dedicated programmer, and in that position runtime was quite important. This helped me to learn that each algorithm involved in a program can have a large impact on its usability in production-grade code. Through that, I found an interest in runtimes of various algorithms, which then led to general efficiency. Since then, I have found that I am particularly interested in sorting algorithms as opposed to any other.

I believe that these two factors are sufficient to qualify me to perform the tasks laid out in the Research Plan.

1. *References*

[1] J. Bentley. (Oct 19, 2000). *Programming Pearls*. (2nd edition). [On-line]. Available: <http://netlib.bell-labs.com/cm/cs/pearls/> [Nov 2, 2013].

[2] R. Sedgewick, J. Bentley. “Quicksort is Optimal.” Internet: <http://www.sorting-algorithms.com/static/QuicksortIsOptimal.pdf>, [Nov 2, 2013].

[3] V. Yaroslavskiy, J. Bloch, J. Bentley. “Replacement of Quicksort in java.util.Arrays with new Dual-Pivot Quicksort”. Available: <http://permalink.gmane.org/gmane.comp.java.openjdk.core-libs.devel/2628>. Personal E-mail (Sep 9-11, 2009).

[4] E. Demaine. Class Lecture, Linear-time Sorting. “Counting Sort, Radix Sort, Lower Bounds for Sorting.” 6.006 Introduction to Algorithms, MIT, Fall 2011.

[5] T. Peters. “Timsort – Python.” Internet: <http://svn.python.org/projects/python/trunk/Objects/listsort.txt>, [Nov 2, 2013].

[6] R. Sedgewick. “Analysis of Shellsort and Related Algorithms.” Internet: <http://www.cs.princeton.edu/~rs/shell/paperF.pdf>, [Nov 2, 2013].

[7] R. Schwartz. “Sorting with the Schwartzian Transform.” Internet: <http://www.stonehenge.com/merlyn/UnixReview/col64.html>, May 2006 [Nov 2, 2013].

[8] J. Katajainen, T. A. Pasanen, “In-place sorting with fewer moves,” Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.3832> [Nov 2 2013].

[9] K. Kiwiel, “Partitioning schemes for quicksort and quickselect,” [arXiv:cs/0312054](http://arxiv.org/abs/cs/0312054) [Nov 2 2013].

[10] R. Muhammad. “Merge Sort.” <http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/mergeSort.htm> [Nov 2 2013].