**Plan of Attack Write Up for The Sorting Challenge**

**Jacob Hull**

**Identifying the Problem**

The problem at hand is, put simply, sorting a larger amount of data than available memory on a machine. In a more in-depth look at the problem, I am being given a virtual machine with two cores of a CPU, one gigabyte of RAM, and about 25 gigabytes of disk space. With these resources, I am tasked with sorting around four gigabytes of thirty-two-bit unsigned integers into ascending order. This requires efficient use of memory and an algorithm that will allow for sorting when all the data is not accessible at the same time. With this stipulation, it will be ever important to keep track of what data is sorted, to what extent, and where it is stored. It will also be very important to find a way to compare data to each other that may be in different files or not in memory. Another issue that may arise is the amount of reads and writes to various files. Knowing this creates the goal of finding an algorithm to sort the data most efficiently all while using the least amount of reads and writes to and from files. So, the overall goal is to find the most efficient way to sort more data than there is available memory, all while writing to and reading from the input, output, and intermediate files the least.

**Identifying Strategies**

The first strategy I thought of was taking as large of a chunk of data that I could fit in memory and sorting a chunk at a time until all the chunks were sorted. Then I would split each chunk in half and sort it with a different chunk until all chunks have been sorted with each other and checked for correctness. To check for correctness, the chunks of sorted data would have a range of data associated with them and I would know that the data is sorted when all the data ranges have no overlaps. I would then put them in the correct order in the output file. The second strategy I thought of was using bubble sort with the same method of grabbing as much data as I can until all is sorted. This would involve multiple iterations over the data to move data to their correct position. The third and final strategy I thought to use after some research online was external sorting. This sorting algorithm is made specifically for handling massive amounts of data. It also works when the data is too big to fit into main memory. To do this, I would divide up the integers into even files and sort them all first using a fast-sorting algorithm. Then, with a pointer to the beginning of each of the files, I would compare all the numbers and put the smallest in the output file and advance the pointer of the file that had the smallest. The process would then repeat until all numbers were compared.

**Evaluate Solutions**

After some deliberation, I found that my first two methods would not be as efficient as using external sort. This is due to the significantly larger number of reads and writes within these two methods. The amount of data touched in these two methods would also be immensely larger than external sort. They would also require many more iterations to completely sort the file and there would be some metadata that would take up extra RAM that could be used during the sorting process. When comparing these two methods to the external sorting method, the external sort shows a decrease in iterations needed to touch the data. This is due to simplifying the sorting into multiple close to RAM sized sections instead of thinking about the data all at once. This method may use more files, as the other two could possibly be done with one or no extra files, but the files will be temporary and will be deleted after the running of the sort. In this project, this will not be an issue, as there is plenty of extra disk space. So, I have decided to use the external sort algorithm to start this project and see what optimizations I can make after I have implemented the main algorithm of the sort.