# Creating a Sorting algorithm, and Stable Sorting
Lab #6

By
Michael Harden

CS 303 Algorithms and Data Structures
September 28, 2021

## 1. Problem Specification

This lab contained two main goals. First, create a algorithm which sorted an array by finding the minimum and maximum elements of a given array, swap them with the start and end index of that array, then repeat this process, shrinking the search space with each pass. For every pass the minimum and maximum elements were moved one index closer to the center. Once the min and max indices corresponded, the whole array was sorted. The second goal was the take a given log file and sort its content with the city names, while keeping the time stamps in order. This required the use of a stable sorting algorithm. I chose to use merge sort to accomplish this goal.

## 2. Program Design

This program requires two classes, Sorts and Driver. The Sorts class contains the implementation for both the Min Max Sorting Algorithm as well as Merge Sort. Merge sort was chosen to sort the log file because it is a stable algorithm which allows for sorting the data based on the city name parameter, while miniating order within the time stamps.

The Driver class contains methods to parse the log files. To sort the logs within a given log file, the Driver class provides a method which utilizes the Merge Sort algorithm from the Sorts class. The Driver class also contains a method to test the implementation of Min Max Sort.

To implement Merge Sort, the same design was used as that in Lab #3

To implement the Min Max Sort the following strategy was employed.

1) Set the current searching space using a left and right pointer. For the first pass this is the entire array. So, the left pointer is set to 0 and right is set to the length of the array - 1.
2) Find the minimum and maximum elements in the searching space
3) Place the minimum element and maximum elements into the first and last indices of the searching space respectively. In doing this step there is 3 possible scenarios.
    a. The maximum element is at the left index and the minimum element is at the right index. In this scenario swap the min and max elements.
    b. The maximum element is at the left index and the minimum element is anywhere other than the right index. In this scenario, first swap the max element with the right index then swap the min element with the new value in the left index.
    c. Any other situation. Swap the min element with the left index, then swap the max element with the right index.
4) Decrease the search space by incrementing the left index by 1 and decrement the right index by 1
5) Continue iterating through this process until the left index is greater than or equal to the right index. At this point the array would be sorted

To parse the files, the Driver class contains the method load_log_file(file_path). This method works in the following way

1) The file passed in to the file_path parameter is opened with the open() method. And its content is read and stored in the content variable with the read() method.
2) Leading and trailing whitespace is stripped away from the content
3) The content is parsed in to an array and split with the delimiter '\n\n'
4) For each line, the last 8 characters on the line are split apart from the line. All the remaining characters are the name of the city.
5) The city and time are placed into a tuple at the index of their line in the array

## 3. Testing

To test the Min Max Sorting algorithm nine test cases were used to attempts and break the sorting algorithm in different ways. Test case #1 was randomly selected positive numbers to verify the algorithm could sort the most basic input. Test #2 was a list of negative numbers. Test #3 was a reverse sorted list. Test #4 alternated in increasing then decreasing numbers. Test #5 contained multiple duplicates. Test #6 was a sorted list. Test #7 was only two elements long and reverse sorted. Test #8 contained one element. And Test #9 was an empty list.

| Test Number | Input | Expected Output |
|---|---|---|
| **#1** | [5, 1, 20, 52, 17, 22, 44, 0, 4, 3] | [ 0, 1, 3, 4, 5, 17, 20, 22, 44, 52] |
| **#2** | [-1, -5, -7, -3, -2, -10] | [-10, -7, -5, -3, -2, -1] |
| **#3** | [10, 9, 8, 7, 6, 5, 4, 3, 2, 1] | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] |
| **#4** | [6, 1, 5, 2, 4, 3] | [1, 2, 3, 4, 5, 6] |
| **#5** | [5, 5, 5, 5, 4, 5, 4] | [4, 4, 5, 5, 5, 5, 5] |
| **#6** | [1, 2, 3, 4, 5, 6] | [1, 2, 3, 4, 5, 6,] |
| **#7** | [1, 0] | [0, 1] |
| **#8** | [0] | |
| **#9** | [] | [] |

To test the way the log files are being sorted, the provided ‘NovelSortInput.txt’ file was used. The sorted out put was as follows:

('Chicago', '09:00:00')

('Chicago', '09:00:59')

('Chicago', '09:03:13')

('Chicago', '09:19:32')

('Chicago', '09:19:46')

('Chicago', '09:21:05')

('Chicago', '09:25:52')

('Chicago', '09:36:14')

('Houston', '09:00:13')

('Houston', '09:01:10')

('Phoenix', '09:00:03')

('Phoenix', '09:14:25')

('Phoenix', '09:37:44')

('Seattle', '09:10:11')

('Seattle', '09:10:25')

('Seattle', '09:22:43')

('Seattle', '09:22:54')

To verify that the algorithm also works for lists containing logs with cities of multiple word names, another file was tested ‘NovelSortingInput_2.txt’. The sorted logs are as follows:

('Birmingham', '09:38:02')

('Birmingham', '09:41:25')

('Birmingham', '09:41:59')

('Chicago', '09:00:00')

('Chicago', '09:00:59')

('Chicago', '09:03:13')

('Chicago', '09:19:32')

('Chicago', '09:19:46')

('Chicago', '09:21:05')

('Chicago', '09:25:52')

('Chicago', '09:36:14')

('Cincinnati', '09:38:45')

('Houston', '09:00:13')

('Houston', '09:01:10')

('Los Angeles', '09:22:01')

('Nashville', '09:38:19')

('New York', '09:10:24')

('New York', '09:42:00')

('Phoenix', '09:00:03')

('Phoenix', '09:14:25')

('Phoenix', '09:37:44')

('Port Saint Joe', '09:41:23')

('Port Saint Joe', '09:42:03')

('Seattle', '09:10:11')

('Seattle', '09:10:25')

('Seattle', '09:22:43')

('Seattle', '09:22:54')

## 4. Analysis and Conclusions

In conclusion, Min Max Sort is not an ideal Sorting Algorithm. With the best, worse, and average case time complexity being O(n^2) there are a lot of better options. One advantage is it is an in-place sort so no extra space would need to be allocated. Another advantage is due to finding both the min and max elements, it would only require half as many iterations as an algorithm such as selection or insertion sort.

For the log file sorting, Merge Sort is a great algorithm to use. Being a stable sorting algorithm, it is possible to sort the logs based on the city name parameter, while maintaining order on the time stamps. Furthermore, having a time complexity of O(n log n) it remains fast as the data size grows. An advantage to the way files are being parsed is, it allows for city with multiple word name and varying word length to also be sorted. Such as ‘New York City’ or ‘Atlantic City’.

## 5. Credits

- The Merge Sort Algorithm was reused from my lab #3 which covers Merge Sort.
- The NovelSortInput.txt was provided on canvas.
- The formatting used on this report is taken from the example report on canvas.