CS470 Lab3 README

Michael Solwold

May 22, 2020

1 Introduction

This README is for a program that uses a user-specified number of threads to sort a given number of integers in a text file. The number of threads, M, are provided via user input and the number of integers to sort, N, are provided via command line argument. For every active thread, it generates a randomly selected (i, j) pair where $i <= j <= N$. That thread will then attempt to retrieve the values from index i to index j. If there is a thread that is currently processing the values from index i to index j, the thread will generate a new (i, j) pair as to not conflict with the work of the first thread.

Once the thread has an (i,j) pair that is not conflicting, it will randomly select a sorting algorithm to sort the values. The possible sorting algorithms are listed below. Once sorted, the thread will write the values back into the file in the same location that it retrieved them from.

A single thread, watcher, will periodically parse the file to check if the file has been sorted. Once the entire file is sorted, the watcher thread will set the global variable 'isSorted' to 1 denoting that the file is sorted and the threads can exit.

2 Requirements

This program is meant to be ran on a **macOS** system. Although, I am not aware of conflicts on other Operating Systems.

3 Instructions

To run this program open a terminal and navigate to the directory where the program Exists. After compiling main.c on a macOS system, run the program from the command line while providing a number of integers for the program to generate.

The program will then prompt the user to provide a number of threads to create.

There are no known limitations given enough time for the program to run its course.

The results will be saved in a text file named ”randomnumbers.txt”.

4 Sorting Algorithms

This program uses QuickSort, Insertion Sort, and Bubble Sort. All three of these algorithms were implemented based on code from GeeksForGeeks.

- Quick Sort: `https://www.geeksforgeeks.org/quick-sort/`
- Insertion Sort: `https://www.geeksforgeeks.org/insertion-sort/`
- Bubble Sort: `https://www.geeksforgeeks.org/bubble-sort/`

5 Mutex Locked Portions of the program

This program utilizes 2 mutex locks:

5.1 fileLock

This lock is required to avoid any conflicts with reading and writing from the file. Unfortunately, this locks the entire file leading to synchronized access to the file.

5.2 sortingLock

This lock handles the acquisition of (i, j) pairs. Every thread will generate an (i, j) pair and then attempt to get the sortingLock. Once the thread has the lock, it references the array described below that keeps track of which areas of the file are being processed. If there are no conflicts, the thread makes the range (i, j) as in-use and then releases the lock. If there are conflicts, the lock is released and the thread generates a new (i, j) pair.

6 Road Blocks

Initially, the lab required region locking of a file. It was recommended that we use fcntl(), a POSIX function that dealt with region locking on files. The goal of this was to have threads sort distinct regions of the file instead of using a mutex lock that would create a synchronous process.

After extensive research into fcntl(), it was discovered that it had not been created with threads in mind. The article linked below discusses this.

`https://www.samba.org/samba/news/articles/low_point/tale_two_stds_os2.html`

There are macros that avoid the issues described in the article, but they were created under the GNU Project and only available to GNU systems.

I was able to implement a very crude version of region locking with an array. If an index was included in a range that was being processed by a thread, its value is 1 in the array. Otherwise, its value is 0. When a thread generates an (i, j) pair, it checks this array for conflicts. If there are none, it sets the range from i to j in the array to 1.

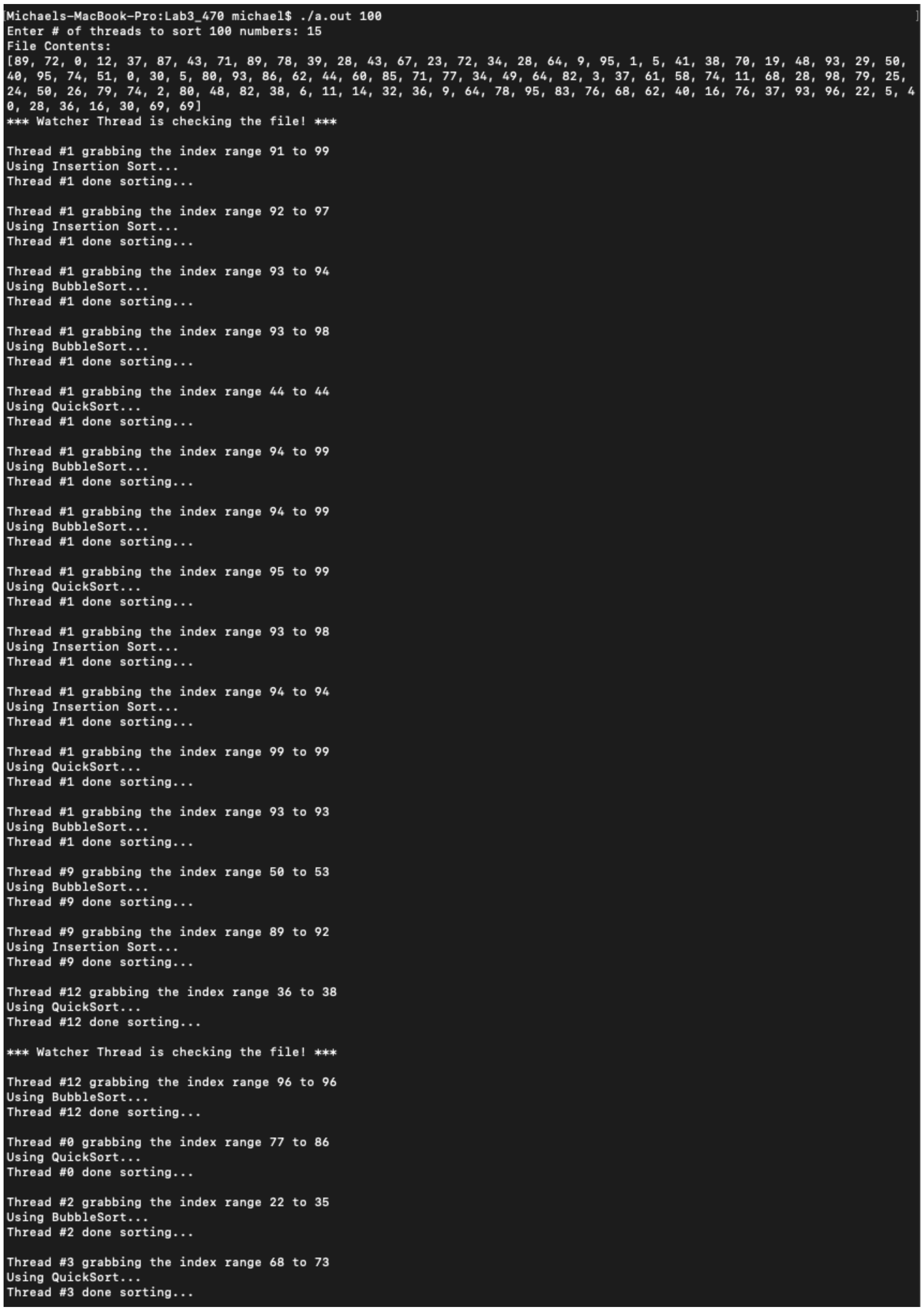
7 Error Handling

This program will exit if an invalid, or missing, number of integers is provided.

Once a valid file is opened, the program will prompt the user for a number of threads. Response should be a positive integer. All other inputs will be rejected and the user will be requested to re-enter their response.

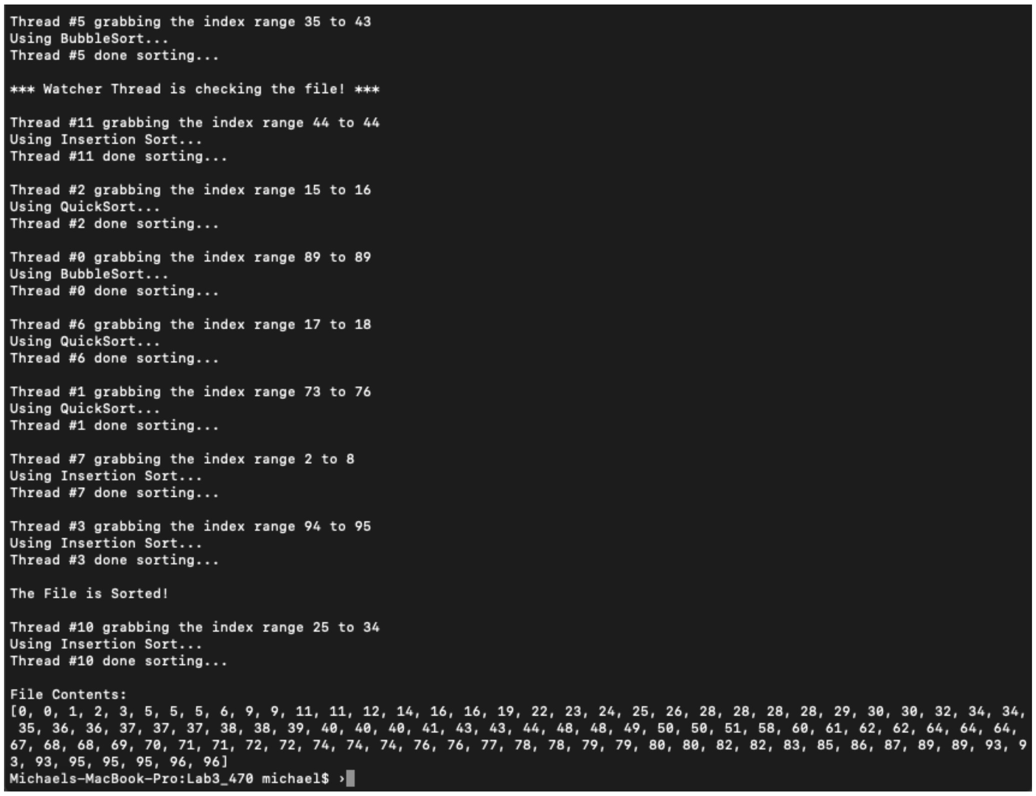
8 Example Output

Due to the program printing out an enormous amount of text, I will provide the start and end of two runs



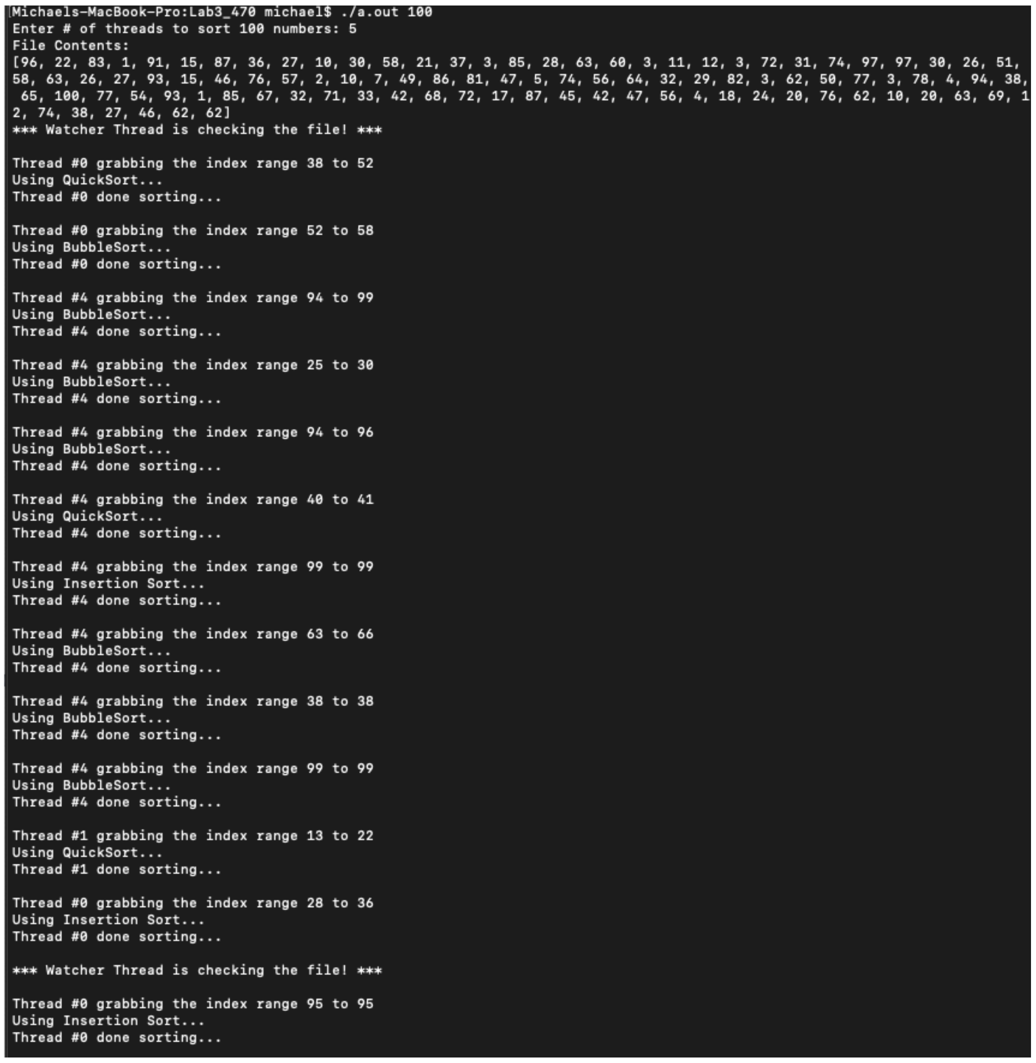
```
Michaels-MacBook-Pro:Lab3_470 michael$ ./a.out 100
Enter # of threads to sort 100 numbers: 15
File Contents:
[89, 72, 0, 12, 37, 87, 43, 71, 89, 78, 39, 28, 43, 67, 23, 72, 34, 28, 64, 9, 95, 1, 5, 41, 38, 70, 19, 48, 93, 29, 50,
40, 95, 74, 51, 0, 30, 5, 80, 93, 86, 62, 44, 60, 85, 71, 77, 34, 49, 64, 82, 3, 37, 61, 58, 74, 11, 68, 28, 98, 79, 25,
24, 50, 26, 79, 74, 2, 80, 48, 82, 38, 6, 11, 14, 32, 36, 9, 64, 78, 95, 83, 76, 68, 62, 40, 16, 76, 37, 93, 96, 22, 5, 4
0, 28, 36, 16, 30, 69, 69]
*** Watcher Thread is checking the file! ***

Thread #1 grabbing the index range 91 to 99
Using Insertion Sort...
Thread #1 done sorting...

Thread #1 grabbing the index range 92 to 97
Using Insertion Sort...
Thread #1 done sorting...

Thread #1 grabbing the index range 93 to 94
Using BubbleSort...
Thread #1 done sorting...

Thread #1 grabbing the index range 93 to 98
Using BubbleSort...
Thread #1 done sorting...

Thread #1 grabbing the index range 44 to 44
Using QuickSort...
Thread #1 done sorting...

Thread #1 grabbing the index range 94 to 99
Using BubbleSort...
Thread #1 done sorting...

Thread #1 grabbing the index range 94 to 99
Using BubbleSort...
Thread #1 done sorting...

Thread #1 grabbing the index range 95 to 99
Using QuickSort...
Thread #1 done sorting...

Thread #1 grabbing the index range 93 to 98
Using Insertion Sort...
Thread #1 done sorting...

Thread #1 grabbing the index range 94 to 94
Using Insertion Sort...
Thread #1 done sorting...

Thread #1 grabbing the index range 99 to 99
Using QuickSort...
Thread #1 done sorting...

Thread #1 grabbing the index range 93 to 93
Using BubbleSort...
Thread #1 done sorting...

Thread #9 grabbing the index range 50 to 53
Using BubbleSort...
Thread #9 done sorting...

Thread #9 grabbing the index range 89 to 92
Using Insertion Sort...
Thread #9 done sorting...

Thread #12 grabbing the index range 36 to 38
Using QuickSort...
Thread #12 done sorting...

*** Watcher Thread is checking the file! ***

Thread #12 grabbing the index range 96 to 96
Using BubbleSort...
Thread #12 done sorting...

Thread #0 grabbing the index range 77 to 86
Using QuickSort...
Thread #0 done sorting...

Thread #2 grabbing the index range 22 to 35
Using BubbleSort...
Thread #2 done sorting...

Thread #3 grabbing the index range 68 to 73
Using QuickSort...
Thread #3 done sorting...
```

Figure 1: Start - Run 1



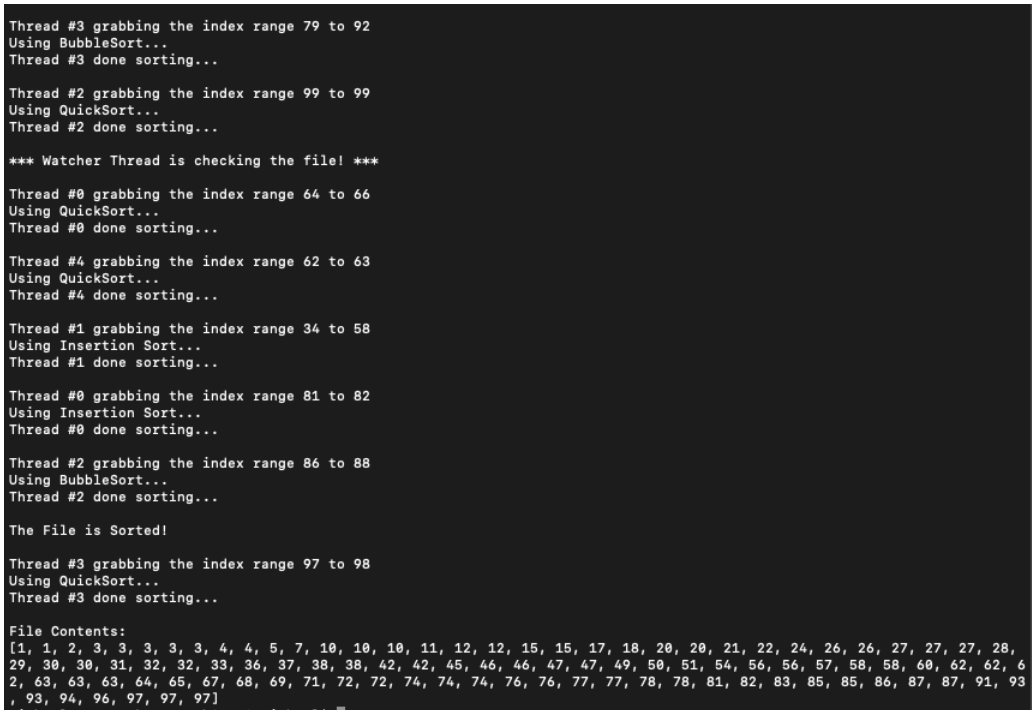
```
Thread #5 grabbing the index range 35 to 43
Using BubbleSort...
Thread #5 done sorting...

*** Watcher Thread is checking the file! ***

Thread #11 grabbing the index range 44 to 44
Using Insertion Sort...
Thread #11 done sorting...

Thread #2 grabbing the index range 15 to 16
Using QuickSort...
Thread #2 done sorting...

Thread #0 grabbing the index range 89 to 89
Using BubbleSort...
Thread #0 done sorting...

Thread #6 grabbing the index range 17 to 18
Using QuickSort...
Thread #6 done sorting...

Thread #1 grabbing the index range 73 to 76
Using QuickSort...
Thread #1 done sorting...

Thread #7 grabbing the index range 2 to 8
Using Insertion Sort...
Thread #7 done sorting...

Thread #3 grabbing the index range 94 to 95
Using Insertion Sort...
Thread #3 done sorting...

The File is Sorted!

Thread #10 grabbing the index range 25 to 34
Using Insertion Sort...
Thread #10 done sorting...

File Contents:
[0, 0, 1, 2, 3, 5, 5, 5, 6, 9, 9, 11, 11, 12, 14, 16, 16, 19, 22, 23, 24, 25, 26, 28, 28, 28, 28, 29, 30, 30, 32, 34, 34,
 35, 36, 36, 37, 37, 37, 38, 38, 39, 40, 40, 40, 41, 43, 43, 44, 48, 48, 49, 50, 50, 51, 58, 60, 61, 62, 62, 64, 64, 64,
67, 68, 68, 69, 70, 71, 71, 72, 72, 74, 74, 74, 76, 76, 77, 78, 78, 79, 79, 80, 80, 82, 82, 83, 85, 86, 87, 89, 89, 93, 9
3, 93, 95, 95, 95, 96, 96]
Michaels-MacBook-Pro:Lab3_470 michael$ >
```

Figure 2: End - Run 1



```
[Michaels-MacBook-Pro:Lab3_470 michael$ ./a.out 100
Enter # of threads to sort 100 numbers: 5
File Contents:
[96, 22, 83, 1, 91, 15, 87, 36, 27, 10, 30, 58, 21, 37, 3, 85, 28, 63, 60, 3, 11, 12, 3, 72, 31, 74, 97, 97, 30, 26, 51,
58, 63, 26, 27, 93, 15, 46, 76, 57, 2, 10, 7, 49, 86, 81, 47, 5, 74, 56, 64, 32, 29, 82, 3, 62, 50, 77, 3, 78, 4, 94, 38,
 65, 100, 77, 54, 93, 1, 85, 67, 32, 71, 33, 42, 68, 72, 17, 87, 45, 42, 47, 56, 4, 18, 24, 20, 76, 62, 10, 20, 63, 69, 1
2, 74, 38, 27, 46, 62, 62]
*** Watcher Thread is checking the file! ***

Thread #0 grabbing the index range 38 to 52
Using QuickSort...
Thread #0 done sorting...

Thread #0 grabbing the index range 52 to 58
Using BubbleSort...
Thread #0 done sorting...

Thread #4 grabbing the index range 94 to 99
Using BubbleSort...
Thread #4 done sorting...

Thread #4 grabbing the index range 25 to 30
Using BubbleSort...
Thread #4 done sorting...

Thread #4 grabbing the index range 94 to 96
Using BubbleSort...
Thread #4 done sorting...

Thread #4 grabbing the index range 40 to 41
Using QuickSort...
Thread #4 done sorting...

Thread #4 grabbing the index range 99 to 99
Using Insertion Sort...
Thread #4 done sorting...

Thread #4 grabbing the index range 63 to 66
Using BubbleSort...
Thread #4 done sorting...

Thread #4 grabbing the index range 38 to 38
Using BubbleSort...
Thread #4 done sorting...

Thread #4 grabbing the index range 99 to 99
Using BubbleSort...
Thread #4 done sorting...

Thread #1 grabbing the index range 13 to 22
Using QuickSort...
Thread #1 done sorting...

Thread #0 grabbing the index range 28 to 36
Using Insertion Sort...
Thread #0 done sorting...

*** Watcher Thread is checking the file! ***

Thread #0 grabbing the index range 95 to 95
Using Insertion Sort...
Thread #0 done sorting...
```

Figure 3: Start - Run 2



```
Thread #3 grabbing the index range 79 to 92
Using BubbleSort...
Thread #3 done sorting...

Thread #2 grabbing the index range 99 to 99
Using QuickSort...
Thread #2 done sorting...

*** Watcher Thread is checking the file! ***

Thread #0 grabbing the index range 64 to 66
Using QuickSort...
Thread #0 done sorting...

Thread #4 grabbing the index range 62 to 63
Using QuickSort...
Thread #4 done sorting...

Thread #1 grabbing the index range 34 to 58
Using Insertion Sort...
Thread #1 done sorting...

Thread #0 grabbing the index range 81 to 82
Using Insertion Sort...
Thread #0 done sorting...

Thread #2 grabbing the index range 86 to 88
Using BubbleSort...
Thread #2 done sorting...

The File is Sorted!

Thread #3 grabbing the index range 97 to 98
Using QuickSort...
Thread #3 done sorting...

File Contents:
[1, 1, 2, 3, 3, 3, 3, 3, 4, 4, 5, 7, 10, 10, 10, 11, 12, 12, 15, 15, 17, 18, 20, 20, 21, 22, 24, 26, 26, 27, 27, 27, 28,
29, 30, 30, 31, 32, 32, 33, 36, 37, 38, 38, 42, 42, 45, 46, 46, 47, 47, 49, 50, 51, 54, 56, 56, 57, 58, 58, 60, 62, 62, 6
2, 63, 63, 63, 64, 65, 67, 68, 69, 71, 72, 72, 74, 74, 74, 76, 76, 77, 77, 78, 78, 81, 82, 83, 85, 85, 86, 87, 87, 91, 93
, 93, 94, 96, 97, 97, 97]
```

Figure 4: End - Run 2