Mitchell Bucklew
#1212442086
CSE 310 T/TH 10:30-11:45

Experiment Report
Design of Experiments

Experiment 1 Sort Algorithm Times:

For testing the execution times of the two different algorithms the testing library I created was not ideal. The issue is that I could not generate a size string of x with this library. Since the library works by generating clusters of characters in strings, it is impossible to generate test cases with character sizes of step 1. So for this test, I created a new function that simply adds a random character to a string on each iteration. In this loop, every iteration will then take this string and encrypt with both merge sort, and insertion sort, recording the runtime of each. This allowed me to create a data set of 1-100 of step one. I found it appropriate to keep this string to one line, as splitting our data into multiple lines will complicate finding the average, opposed to simply putting all the test data into one line and running the encryption function once.

To collect this data, I went ahead and timed the execution of each function, appended them to a string in the form of 'x, y1, y2\n' and was able to copy and paste this format into Desmos to be graphed. At first, the data had weird variations and there were clear outliers. I changed the code to take an average of 4 executions for each set.

```
auto start = high_resolution_clock::now();
auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);
```

Experiment 2 Compression Ratio :

I will analyze the compression ratio to find out how the algorithm changes when encrypting several lines at a time. #4 on the project 1 description gave me the inspiration to try this out. To conduct the experiment I will collect all the statistical data from the compression ratio and compare them with similar data of multiple line encryption.

Experiment 3 Repeat Character Count:

To test how the algorithm compression ratio works, a different test needed to be designed than the sort execution time tests. For this, I created a test library (testlibrary.h), which will generate bodies of strings based on a few input parameters. Included are strings per line, amount of lines, amount of characters that repeat, and amount of times each character repeats. I thought this would be useful for testing which characteristics of strings will allow for higher compression ratios. Knowing the encoding algorithm, the more repeated the characters the higher the compression ratio. I will change these parameters to try and get the highest compression ratio possible. This test library generates strings of 10-14 characters long then concats them between spaces to mimic words. Actual example output:

Generate one like, with two strings on each. Each string will have one character that repeats 3 times

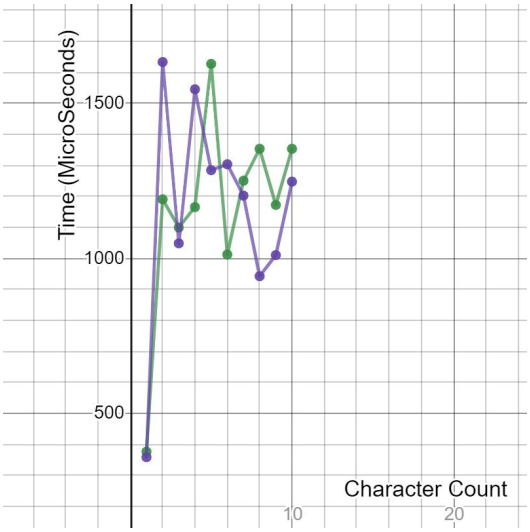
Mitchell Bucklew
#1212442086
CSE 310 T/TH 10:30-11:45

Experiment Report

`test.generate(1, 2, 1, 3)` = "FVWYZWW CGBEMTCDC \n"

Experiment 1

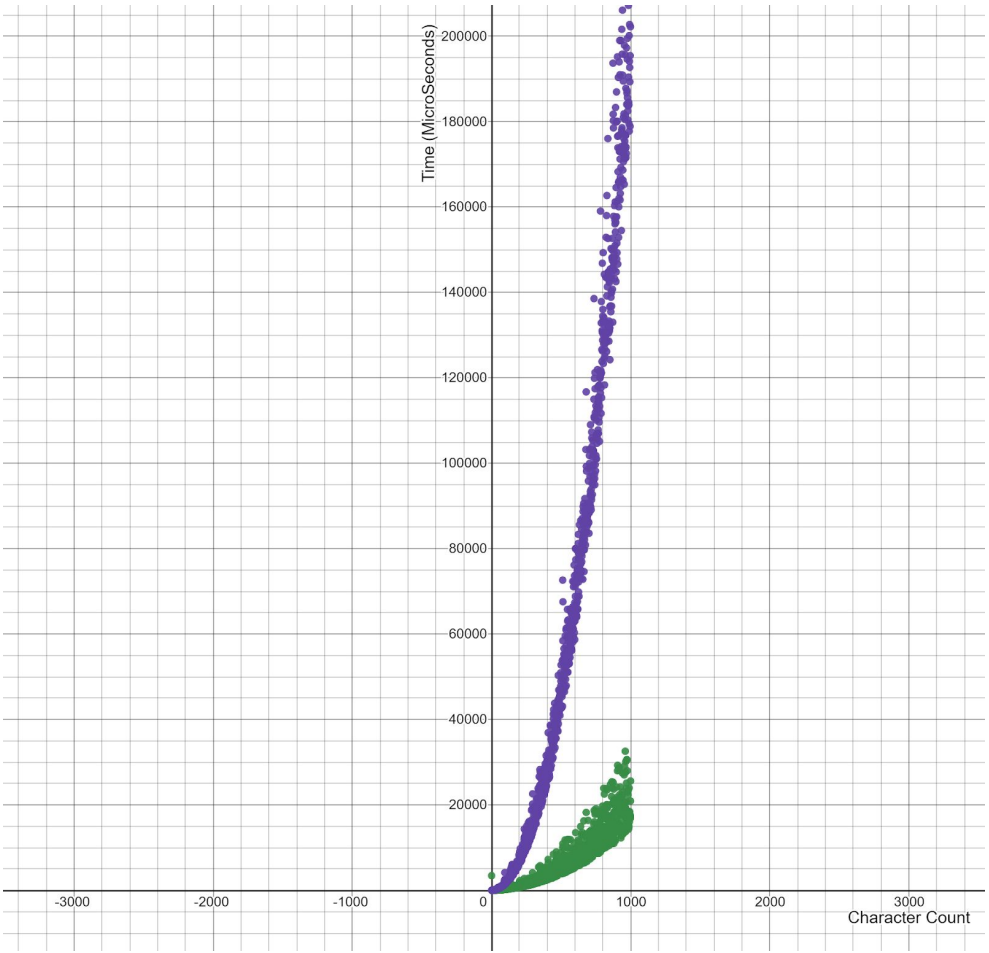
Insertion Sort Vs MergeSort 10 items



- Merge sort
- Insertion

Points:

1, 376, 358
2, 1190, 1633
3, 1099, 1048
4, 1165, 1545
5, 1627, 1284
6, 1012, 1303
7, 1250, 1202
8, 1353, 942
9, 1172, 1010
10, 1353, 1247



Insertion Sort Vs MergeSort 1000 Items (Each value is an average of 10 executions)

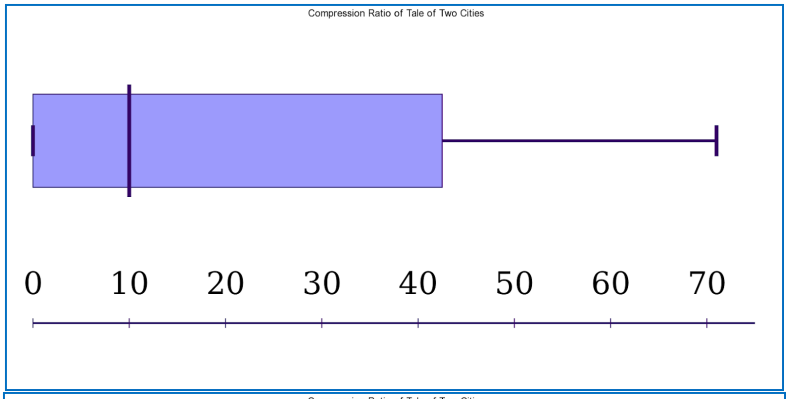
- Merge sort
- Insertion

Mitchell Bucklew
#1212442086
CSE 310 T/TH 10:30-11:45

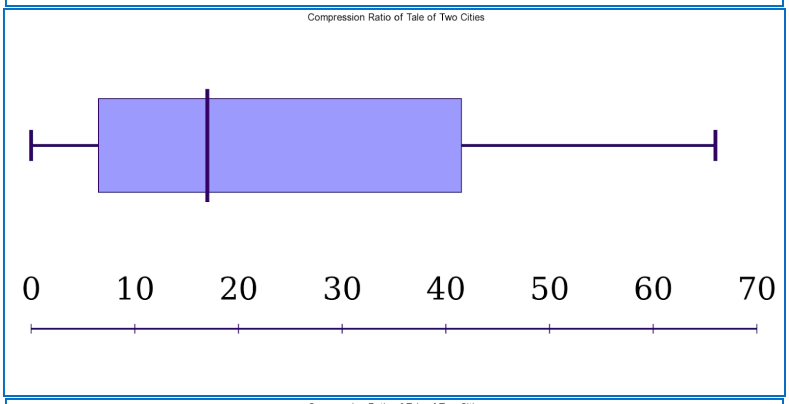
Experiment Report

Experiment 2

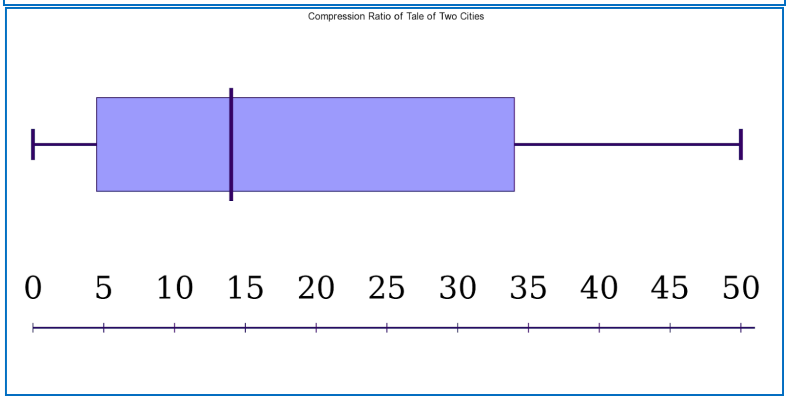
Encoding “Tale of Two Cities” in Multiple Lines



1 Line at a time
Min=0
Max = 71
Median = 10
Lower Median = 0
Upper Median = 14
Mean = 9.477
Standard Deviation = 7.8167



2 Lines at a time
Min=0
Max = 50
Median = 14
Lower Median = 9
Upper Median = 18
Mean = 14.09
Standard Deviation = 7.97



3 Lines at a time
Min=0
Max= 66
Median = 10
Lower Median = 17
Upper Median = 17
Mean = 16.94

Mitchell Bucklew
#1212442086
CSE 310 T/TH 10:30-11:45

Experiment Report
Standard Deviation = 7.00

Experiment 3

test1.generate(line count, string counter, amount of characters that repeat, amount of times characters repeat)

//Generates strings with no common characters
test1.generate(50, 50, 1, 1);
Averages => 4.3,4.36,4.24,4.51 = 4.3525

//Generations strings with one character that repeats 4 times, example AAAAERGINJDFG
test1.generate(50, 50, 1, 4);
Averages => 5.52, 5.56,5.4,5.42,5.36 = 5.452

test1.generate(50, 50, 2, 4);
Averages => 19.73, 20.34,19.86,20.2 = 20.0325

test1.generate(50, 50, 2, 5);
Averages => 38.04, 37.82,37.82,36.84,37.47 = 37.598

Mitchell Bucklew
#1212442086
CSE 310 T/TH 10:30-11:45

Experiment Report

Conclusion

In experiment 1, I analyzed the running times of Merge Sort vs Insertion Sort. My findings showed Merge sort to be much faster than insertion, especially for longer input size. I also found that merge sort can be faster than insertion, but only for very small character sizes of < 5.

In experiment 2 I gathered statistical data on the compression ratio using the tale of two cities given to us in sample inputs. I compared data from encoding 1, to 2, to 3 lines at a time. Analyzing the data showed me the compression ratio goes up the higher the input size is. This tells if, to save the most space encode as much data at one time as possible. This makes sense because of the more repeating characters, the higher the cluster count and higher the ratio.

In experiment 3 I tested how the repeat of characters within strings influences the compression ratio. I confirmed my hypothesis, as the compression ratio increases greatly as the repeat of characters within strings repeat as well. This means some data will work much better with this algorithm than others, depending on how many repeated characters there is on each input.