Project 3

Vincent Taylor, Harsh Patel Systems Programming

Problem Statement :

Sort a given CSV file of variable size by a column entered by the user. When given a working directory the program should locate all csv files and sort them according to the given sorting parameter. Place all the final sorted components into a single file renamed for the user. If the user specifies a given output directory output the final CSV there. If the desired output directory does not exist create it; if no output directory is given output the final CSV to a default directory.

Approach :

Recursively go through each directory and file of the given directory sort all the files and store them within a single data structure resort them and place them within the final single CSV file in the specified directory.

Tests / Methodology:

Described below are some of the design implementations and possible implementations. Each will provide a paper analysis, basic system test, moderate system test, and advanced system test each defined in the computer specs section of this document (Note these three machines will vary in part performance, capacity, and efficiency. Each test on the system parts will run through several dozen randomly generated test cases. These test cases are pre created and consistent among each of the systems the given quantity of files, directories, and average number of lines will be included in the test case section.The given number of test cases will be 10 for now, but may expand. Each timing mentioned below will use the time command in the unix shell to get real time in secs to test efficiency of each method. Time will be measured in clock seconds.

1) Sorting : Assuming Worst Case

Paper Analysis;

Bubble Sort :

Time : O(n^2)

Space : O(n)

Quicksort :

Time : O(n log n)

Space : O(n)

Bubble/Quicksort :

Time : O(kn)

Space : O(n)

Run Time Analysis :

Quicksort	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Basic	1.049	0.039	0.665	1.706	0.005	1.229	0.373	0.139	0.411	0.607
Moderate	0.532	0.024	0.343	0.894	0.009	0.629	0.192	0.078	0.216	0.502
Advanced	0.431	0.022	0.295	0.789	0.003	0.735	0.223	0.059	0.210	0.500

Bubble Sort	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Basic	31.332	0.47	18.942	52.595	0.008	37.173	9.738	2.967	11.019	29.22
Moderate	16.078	0.237	9.686	27.078	0.009	19.096	4.96	1.495	5.657	14.95
Advanced	14.98	0.138	10.21	22.335	0.009	20.105	2.98	1.455	5.032	11.45

Quicksort /Bubble	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Basic	0.757	0.032	0.478	1.235	0.004	0.89	0.263	0.106	0.302	0.723
Moderate	0.389	0.02	0.245	0.634	0.009	0.457	0.137	0.054	0.154	0.364
Advanced	0.0224	0.02	0.235	0.645	0.009	0.325	0.12	0.05	0.133	0.297

2) File Detection

Paper Analysis;

Multi-Threading :

Should be faster based on the use of parallelism so that all the directories Could be searched while the files are sent in

Forking :

Potentially slower, since it is a whole separate program that is spawned every time there is a new directory or file found.

Run Time Analysis:

Multi-Threading	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Basic	0.0970	0.0350	0.0380	0.0700	0.0160	0.0870	0.0550	0.0470	0.0600	0.0530
Moderate	0.0900	0.0250	0.0390	0.0580	0.0220	0.0820	0.0560	0.0440	0.0570	0.0570
Advanced	0.0060	0.0040	0.0040	0.0050	0.0030	0.0050	0.0050	0.0040	0.0050	0.0040

Forking	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Basic	0.2060	0.0450	0.0740	0.1430	0.0160	0.1430	0.0940	0.0690	0.1050	0.0950
Moderate	0.1530	0.0420	0.0640	0.1140	0.0620	0.1040	0.0970	0.0630	0.0850	0.0770
Advanced	0.0380	0.0060	0.0100	0.0140	0.0040	0.0250	0.0140	0.0150	0.0180	0.0110

3) File Combination :

Paper Analysis;

Single file:

This should be faster by reading them into one file and sorting it as a whole

Temp into Final:

Could potentially be slower because we have varying read and write times. Plus disk speed is slower than computation speed.

Runtime Analysis:

Single File	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Basic	2.14	1.69	2.735	2.561	0.052	3.145	2.16	1.932	3.15	5.05
Moderate	2.01	1.59	2.104	2.934	0.033	2.94	1.99	1.783	2.94	4.123
Advanced	1.787	1.29	1.56	1.632	0.032	2.56	2.31	1.882	2.3	3.87
Temp into File	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Basic	10.87	2.45	3.893	3.45	0.07	2.66	2.1	2.2	2.45	4.321
Moderate	9.34	2.101	3.1	3.16	0.064	2.562	1.88	1.67	2.134	3.23
Advanced	6.89	2.3	3.453	2.86	0.05	2.145	2.3	1.39	1.983	2.789

Computer Specs :

Basic :

Cpu: i3 - 26000u
Ram: 2GB DDR2
Storage: 120 GB 4800 rpm
Graphics : Integrated
OS : Windows

Moderate :

Cpu : i5 - 73000HQ
Ram : 8GB DDR3
Storage : 1Tb HDD 7200 rpm
Graphics : GTX 1050
OS : Windows

Advanced :

Cpu: i7 - 8770k (4.5Ghz)
Ram: 16 GB DDR3

Storage: 256 Gb SSD
Graphics : GTX 780 OC
OS : Linux

Test Cases :

1)
Files : 151
Lines (Avg) : 3150
Directories : 22
2)
Files : 19
Lines (Avg) : 400
Directories : 4
3)
Files : 45
Lines (Avg) : 2456
Directories : 5
4)
Files : 64
Lines (Avg) : 4080
Directories : 8
5)
Files : 2
Lines (Avg) : 5037
Directories : 1
6)
Files : 102
Lines (Avg) : 3429
Directories : 14
7)
Files : 63
Lines (Avg) : 1765
Directories : 9
8)
Files : 38
Lines (Avg) : 986
Directories : 9
9)

Files : 67
Lines (Avg) : 1879
Directories : 10

10)

Files : 43
Lines (Avg) : 3044
Directories : 10

Conclusion :

From this well can clearly see that quicksort with bubble sort is much faster than bubble sort and moderately faster than just quicksort. Additionally, through testing and trials Harsh and I were able to deduce that a k with 1/20 the size of the array was ideal for this quicksort/bubble sort combination. Next up well had to decide between using a temp file for our one huge file sort or making a single array structure to fill them. Through the data well can see the machines with HDD's perform moderately slower; However, the inclusion of an SSD does speed up performance, but it still lags behind the single struct sort method. When looking at the method for file discovery, multi-threading is significantly better than forking through the directories. These results showed us that for the fastest result well should multithreaded through our file and directories and place them into a single struct and sort them via a bubble/quicksort combination