Final Design Document
Parallel Thread Sorting

All files description:
Makefile: Helpers make and clean the program files

ParallelSorting.c: The file in which the project is done in. It randomly fills an array with ints, and tries various array sizes, split sizes, and sorting techniques. All combinations are outputted.

ParallelSorting: The executable after calling “make”

ParallelSorting.slurm: The slurm file to run on the HPC

changesDoc.txt: The program changes between the beta version and the final version

designDoc.txt: The old design document for the beta version

programOutput.txt: The output of my program when compiled and run, and this output was piped to this file.

ProofOfCorrectness.txt: The output of my program, but one of my printf() statements are un-commented. This file shows the sorted arrays for one of the cases in my code, just to show that my program is in fact sorting random arrays correctly using different split sizes and types of sorting algorithms.

Major data structures:
POSIX threads: to parallelize the sorting
A simple mutex semaphore: To control the printf() flow, which is commented out in my final version of the program.

Progress:
My progress report are my git commits:
https://github.com/AdamKBeck/Parallel-Thread-Sorting/commits/master

Output:
Here is the output of ProofOfCorrectness.txt after running my program on my computer. Again, it shows the output for my final version, as well as some printf() statements to show the the arrays are in fact sorted.

Solution indices after sorting: [304089172, 424238335, 596516649, 719885386, 783368690, 846930886, 1025202362, 1102520059, 1189641421, 1303455736, 1350490027, 1365180540, 1540383426, 1649760492, 1681692777, 1714636915, 1804289383, 1957747793, 1967513926, 2044897763,]

Finished!
Array size 20 with 2 threads and choice 1 took 0.000107 seconds
Solution indices after sorting: [35005211, 233665123, 278722862, 294702567, 336465782, 468703135, 521595368, 628175011, 635723058, 861021530, 1059961393, 1101513929, 1125898167, 1315634022, 1369133069, 1656478042, 1726956429, 1801979802, 2089018456, 2145174067,]

Finished!
Array size 20 with 2 threads and choice 2 took 0.000030 seconds
Solution indices after sorting: [42999170, 135497281, 137806862, 149798315, 184803526, 412776091, 608413784, 749241873, 756898537, 859484421, 982906996, 1129566413, 1131176229, 1424268980, 1653377373, 1734575198, 1911759956, 1914544919, 1973594324, 2038664370,]

Finished!
Array size 20 with 2 threads and choice 3 took 0.000025 seconds
Solution indices after sorting: [84353895, 511702305, 572660336, 610515434, 760313750, 805750846, 939819582, 1100661313, 1141616124, 1159126505, 1374344043, 1433925857, 1548233367, 1585990364, 1632621729, 1827336327, 1937477084, 1998898814, 2001100545, 2084420925,]

Finished!
Array size 20 with 4 threads and choice 1 took 0.000086 seconds
Solution indices after sorting: [356426808, 491705403, 709393584, 752392754, 855636226, 943947739, 945117276, 1264095060, 1411549676, 1469348094, 1474612399, 1477171087, 1749698586, 1780695788, 1843993368, 1889947178, 1918502651, 1956297539, 1984210012, 2053999932,]

Finished!
Array size 20 with 4 threads and choice 2 took 0.000040 seconds
Solution indices after sorting: [317097467, 364228444, 402724286, 463480570, 485560280, 593209441, 603570492, 660260756, 894429689, 927612902, 959997301, 1036140795,

1194953865, 1330573317, 1376710097, 1687926652, 1892066601, 1947346619, 1975960378, 2040651434,]

Finished!
Array size 20 with 4 threads and choice 3 took 0.000038 seconds
Solution indices after sorting: [221558440, 269455306, 270744729, 327254586, 352406219, 498777856, 524872353, 631704567, 791698927, 822890675, 1063958031, 1255179497, 1469834481, 1572276965, 1600028624, 1610120709, 1633108117, 1703964683, 2007905771, 2114738097,]

Finished!
Array size 20 with 8 threads and choice 1 took 0.000159 seconds
Solution indices after sorting: [112805732, 150122846, 160051528, 168002245, 200747796, 289700723, 378409503, 439493451, 515530019, 990892921, 1117142618, 1120048829, 1373226340, 1409959708, 1573363368, 1631518149, 1713258270, 1760243555, 2040332871, 2077486715,]

Finished!
Array size 20 with 8 threads and choice 2 took 0.000168 seconds
Solution indices after sorting: [8936987, 111537764, 116087764, 155324914, 269441500, 338888228, 350322227, 387346491, 438792350, 841148365, 1231192379, 1275373743, 1622597488, 1760281936, 1869470124, 1911165193, 1960709859, 1982275856, 2142757034, 2147469841,]

Finished!
Array size 20 with 8 threads and choice 3 took 0.000084 seconds
Solution indices after sorting: [76065818, 213975407, 653468858, 771151432, 971899228, 1139901474, 1186452551, 1239036029, 1244316437, 1350573793, 1476153275, 1605894428, 1605908235, 1626276121, 1784639529, 1789366143, 1875335928, 1884661237, 1987231011, 2130794395,]

Finished!
Array size 20 with 16 threads and choice 1 took 0.000604 seconds
Solution indices after sorting: [159259470, 165344818, 352118606, 492067917, 496987743, 532670688, 680466996, 706043324, 1067854538, 1351797369, 1359512183, 1395235128, 1432114613, 1504569917, 1597322404, 1782436840, 1909002904, 1939964443, 2103318776, 2112255763,]

Finished!

Array size 20 with 16 threads and choice 2 took 0.000558 seconds
Solution indices after sorting: [6939507, 243268139, 480298490, 502278611, 601385644, 722308542, 740759355, 933110197, 968338082, 1012502954, 1096689772, 1172755590, 1272469786, 1285228804, 1398295499, 1544617505, 1789376348, 1820388464, 2027907669, 2086206725,]

Finished!
Array size 20 with 16 threads and choice 3 took 0.000222 secondsFinished!
Array size 200 with 2 threads and choice 1 took 0.000024 secondsFinished!
Array size 200 with 2 threads and choice 2 took 0.000034 secondsFinished!
Array size 200 with 2 threads and choice 3 took 0.000021 secondsFinished!
Array size 200 with 4 threads and choice 1 took 0.000035 secondsFinished!
Array size 200 with 4 threads and choice 2 took 0.000037 secondsFinished!
Array size 200 with 4 threads and choice 3 took 0.000034 secondsFinished!
Array size 200 with 8 threads and choice 1 took 0.000115 secondsFinished!
Array size 200 with 8 threads and choice 2 took 0.000119 secondsFinished!
Array size 200 with 8 threads and choice 3 took 0.000115 secondsFinished!
Array size 200 with 16 threads and choice 1 took 0.000732 secondsFinished!
Array size 200 with 16 threads and choice 2 took 0.000626 secondsFinished!
Array size 200 with 16 threads and choice 3 took 0.000419 secondsFinished!
Array size 500 with 2 threads and choice 1 took 0.000034 secondsFinished!
Array size 500 with 2 threads and choice 2 took 0.000035 secondsFinished!
Array size 500 with 2 threads and choice 3 took 0.000040 secondsFinished!
Array size 500 with 4 threads and choice 1 took 0.000065 secondsFinished!
Array size 500 with 4 threads and choice 2 took 0.000064 secondsFinished!
Array size 500 with 4 threads and choice 3 took 0.000063 secondsFinished!
Array size 500 with 8 threads and choice 1 took 0.000189 secondsFinished!
Array size 500 with 8 threads and choice 2 took 0.000220 secondsFinished!
Array size 500 with 8 threads and choice 3 took 0.000168 secondsFinished!
Array size 500 with 16 threads and choice 1 took 0.001863 secondsFinished!
Array size 500 with 16 threads and choice 2 took 0.001772 secondsFinished!
Array size 500 with 16 threads and choice 3 took 0.000981 secondsFinished!
Array size 2500 with 2 threads and choice 1 took 0.008658 secondsFinished!
Array size 2500 with 2 threads and choice 2 took 0.006368 secondsFinished!
Array size 2500 with 2 threads and choice 3 took 0.000153 secondsFinished!
Array size 2500 with 4 threads and choice 1 took 0.001511 secondsFinished!
Array size 2500 with 4 threads and choice 2 took 0.001004 secondsFinished!
Array size 2500 with 4 threads and choice 3 took 0.000281 secondsFinished!
Array size 2500 with 8 threads and choice 1 took 0.000403 secondsFinished!

Array size 2500 with 8 threads and choice 2 took 0.001033 secondsFinished!
Array size 2500 with 8 threads and choice 3 took 0.000462 secondsFinished!
Array size 2500 with 16 threads and choice 1 took 0.001816 secondsFinished!
Array size 2500 with 16 threads and choice 2 took 0.001262 secondsFinished!
Array size 2500 with 16 threads and choice 3 took 0.000696 secondsFinished!
Array size 10000 with 2 threads and choice 1 took 0.049377 secondsFinished!
Array size 10000 with 2 threads and choice 2 took 0.100876 secondsFinished!
Array size 10000 with 2 threads and choice 3 took 0.001081 secondsFinished!
Array size 10000 with 4 threads and choice 1 took 0.030729 secondsFinished!
Array size 10000 with 4 threads and choice 2 took 0.044802 secondsFinished!
Array size 10000 with 4 threads and choice 3 took 0.000604 secondsFinished!
Array size 10000 with 8 threads and choice 1 took 0.018646 secondsFinished!
Array size 10000 with 8 threads and choice 2 took 0.025936 secondsFinished!
Array size 10000 with 8 threads and choice 3 took 0.001312 secondsFinished!
Array size 10000 with 16 threads and choice 1 took 0.005336 secondsFinished!
Array size 10000 with 16 threads and choice 2 took 0.012201 secondsFinished!
Array size 10000 with 16 threads and choice 3 took 0.002283 seconds

HPC output:

Below is the output on the HPC. I believe that it runs my program so fast that the timer shows “0 seconds” for most runs. It makes sense, as sorting an array of size 20, for example, should take next to no time on the HPC. When my array is of size 10,000 you can see that the HPC does in fact start to take a little bit of time to sort it. As such, I choose to graph my output from my computer, as listed above.

gcc -o ParallelSorting ParallelSorting.c -lpthread -lrt -std=c99
Finished!
Array size 20 with 2 threads and choice 0 took 0.000000 secondsFinished!
Array size 20 with 2 threads and choice 1 took 0.000000 secondsFinished!
Array size 20 with 2 threads and choice 2 took 0.000000 secondsFinished!
Array size 20 with 4 threads and choice 0 took 0.000000 secondsFinished!
Array size 20 with 4 threads and choice 1 took 0.000000 secondsFinished!
Array size 20 with 4 threads and choice 2 took 0.000000 secondsFinished!
Array size 20 with 8 threads and choice 0 took 0.000000 secondsFinished!
Array size 20 with 8 threads and choice 1 took 0.000000 secondsFinished!
Array size 20 with 8 threads and choice 2 took 0.000000 secondsFinished!
Array size 20 with 16 threads and choice 0 took 0.000000 secondsFinished!
Array size 20 with 16 threads and choice 1 took 0.000000 secondsFinished!

Array size 20 with 16 threads and choice 2 took 0.000000 secondsFinished!
Array size 200 with 2 threads and choice 0 took 0.000000 secondsFinished!
Array size 200 with 2 threads and choice 1 took 0.000000 secondsFinished!
Array size 200 with 2 threads and choice 2 took 0.000000 secondsFinished!
Array size 200 with 4 threads and choice 0 took 0.000000 secondsFinished!
Array size 200 with 4 threads and choice 1 took 0.000000 secondsFinished!
Array size 200 with 4 threads and choice 2 took 0.000000 secondsFinished!
Array size 200 with 8 threads and choice 0 took 0.000000 secondsFinished!
Array size 200 with 8 threads and choice 1 took 0.000000 secondsFinished!
Array size 200 with 8 threads and choice 2 took 0.000000 secondsFinished!
Array size 200 with 16 threads and choice 0 took 0.000000 secondsFinished!
Array size 200 with 16 threads and choice 1 took 0.000000 secondsFinished!
Array size 200 with 16 threads and choice 2 took 0.000000 secondsFinished!
Array size 500 with 2 threads and choice 0 took 0.000000 secondsFinished!
Array size 500 with 2 threads and choice 1 took 0.010000 secondsFinished!
Array size 500 with 2 threads and choice 2 took 0.000000 secondsFinished!
Array size 500 with 4 threads and choice 0 took 0.000000 secondsFinished!
Array size 500 with 4 threads and choice 1 took 0.000000 secondsFinished!
Array size 500 with 4 threads and choice 2 took 0.000000 secondsFinished!
Array size 500 with 8 threads and choice 0 took 0.000000 secondsFinished!
Array size 500 with 8 threads and choice 1 took 0.000000 secondsFinished!
Array size 500 with 8 threads and choice 2 took 0.000000 secondsFinished!
Array size 500 with 16 threads and choice 0 took 0.000000 secondsFinished!
Array size 500 with 16 threads and choice 1 took 0.000000 secondsFinished!
Array size 500 with 16 threads and choice 2 took 0.000000 secondsFinished!
Array size 2500 with 2 threads and choice 0 took 0.000000 secondsFinished!
Array size 2500 with 2 threads and choice 1 took 0.000000 secondsFinished!
Array size 2500 with 2 threads and choice 2 took 0.010000 secondsFinished!
Array size 2500 with 4 threads and choice 0 took 0.000000 secondsFinished!
Array size 2500 with 4 threads and choice 1 took 0.000000 secondsFinished!
Array size 2500 with 4 threads and choice 2 took 0.010000 secondsFinished!
Array size 2500 with 8 threads and choice 0 took 0.000000 secondsFinished!
Array size 2500 with 8 threads and choice 1 took 0.000000 secondsFinished!
Array size 2500 with 8 threads and choice 2 took 0.000000 secondsFinished!
Array size 2500 with 16 threads and choice 0 took 0.000000 secondsFinished!
Array size 2500 with 16 threads and choice 1 took 0.000000 secondsFinished!
Array size 2500 with 16 threads and choice 2 took 0.010000 secondsFinished!
Array size 10000 with 2 threads and choice 0 took 0.000000 secondsFinished!
Array size 10000 with 2 threads and choice 1 took 0.060000 secondsFinished!

Array size 10000 with 2 threads and choice 2 took 0.130000 secondsFinished!
Array size 10000 with 4 threads and choice 0 took 0.000000 secondsFinished!
Array size 10000 with 4 threads and choice 1 took 0.030000 secondsFinished!
Array size 10000 with 4 threads and choice 2 took 0.070000 secondsFinished!
Array size 10000 with 8 threads and choice 0 took 0.000000 secondsFinished!
Array size 10000 with 8 threads and choice 1 took 0.020000 secondsFinished!
Array size 10000 with 8 threads and choice 2 took 0.030000 secondsFinished!
Array size 10000 with 16 threads and choice 0 took 0.000000 secondsFinished!
Array size 10000 with 16 threads and choice 1 took 0.020000 secondsFinished!
Array size 10000 with 16 threads and choice 2 took 0.020000 seconds

Graph:
Since I am changing 4 variables, I will need a 4D graph (time vs. sorting choice vs. slice count vs. array size).

Instead, I will graph the mean time vs. array size for each of my 3 sorting choices (quicksort, insertion sort, and bubble sort).

As you can see below, quicksort was by far the fastest method of sorting. It did not blow up in time compared to insertion sort and bubble sort. All 3 sorting methods very fairly similar for array sizes 20, 200, and 500, as the sizes just aren't big enough for a different sorting method to dominate the runtime. Instead, the runtime at these small sizes depends mostly on the speed at which my slices are merged together, and the speed at which the slices are calculated, and the threads are created and joined.

2		
3	20	0.000615
4	200	0.00047025
5	500	0.00059075
6	2500	0.000642
7	10000	0.0212155
8		
9		
10		
11		
12	Insertion-sort	
13	20	0.000491
14	200	0.00045075
15	500	0.0006205
16	2500	0.00220875
17	10000	0.0496545
18		
19	Quick-sort	
20	20	0.000247
21	200	0.0002845
22	500	0.00029725
23	2500	0.0004885
24	10000	0.00117325
25		
26		
27		
28		
29		
30		
31		
32		
33		
34		
35		
36		
37		
38		
39