Benjamin Cohen

**Experiment:**

Data Size: 645288 strings

For each trial, I ran each implementation 50 times, averaging the last 30.

Merge – Sedgewick's Merge from the textbook

MergeX – Sedgewick's Merge with a cutoff for insertion sort and checking to see if the subarrays are sorted already before merging

JDK – Arrays.parallelsort() which is the java library function for parallel sorting.

/*Parallel – My implementation of Merge with threading. I create the threads during the sort down phase when breaking up the array if it is below 100000. Each thread has its own lo and hi variables so there is no conflict between cooperative threads. Each thread is responsible for sorting between their own lo and hi. The threads do this by forking 2 threads, leftPart and rightPart. The left part spans from 0 to mid and the right part goes from mid+1 to hi. Each of those threads in turn are responsible for sorting their lo, hi.

By calling join on the left and right part, the parent thread knows that its children threads, leftPart and rightPart have finished their job; (meaning left and right are now sorted). It can therefore safely call merge with leftPart and rightPart. This merge function is the same as the Merge part in the Merge function. It works because the only thing needed for merge to work is for leftPart and rightPart to be sorted which has to be true if the left thread and the right thread both terminated (by in turn waiting for their threads to finish sorting their subarray and merging those). This thread spawning and waiting for other threads to finish, and merging when both leftPart and rightPart are finished is done in parallel, making it faster.

I think there is room to optimize this by breaking up the array into more localized chunks. It seems weird that when I'm sorting I'm doing things on opposites sides of the array. It might make more sense to split up the array into 16 parts and only send out the left 8 parts at a time and wait for those to return (and merge those up) before doing the right half, that way everything is more local.

NOTES (Data on next page)

1. Input was randomized without a seed. I thought it would be more useful to see if mergex was functioning more properly since it should have somewhat of a higher standard deviation as it checks to see if the array is in order before calling merge which may be more variant if the array is ordered differently.
2. I had 4 processors so it's not overly surprising that Merge/JDK and Parallel/JDK ratio to Parallel is different compared to the sample results in the handout. (Parallel is slower) Since the time of Parallel is closer to JDK than Merge I don't think it's a program bug.

| | Merge | MergeX | JDK | Parallel | Merge/JDK | MergeX/JDK | JDK/JDK | Parallel/JDK |
|---|---|---|---|---|---|---|---|---|
| 1 | 392 | 379 | 160 | 211 | 2.45 | 2.36875 | 1 | 1.31875 |
| 2 | 411 | 392 | 167 | 207 | 2.461078 | 2.347305 | 1 | 1.239521 |
| 3 | 396 | 381 | 161 | 209 | 2.459627 | 2.36646 | 1 | 1.298137 |
| 4 | 408 | 394 | 168 | 211 | 2.428571 | 2.345238 | 1 | 1.255952 |
| 5 | 409 | 395 | 166 | 212 | 2.463855 | 2.379518 | 1 | 1.277108 |
| 6 | 427 | 393 | 167 | 216 | 2.556886 | 2.353293 | 1 | 1.293413 |
| 7 | 409 | 393 | 167 | 211 | 2.449102 | 2.353293 | 1 | 1.263473 |
| 8 | 403 | 390 | 166 | 214 | 2.427711 | 2.349398 | 1 | 1.289157 |
| 9 | 404 | 388 | 174 | 244 | 2.321839 | 2.229885 | 1 | 1.402299 |
| 10 | 419 | 391 | 165 | 210 | 2.539394 | 2.369697 | 1 | 1.272727 |
| 11 | 397 | 389 | 165 | 215 | 2.406061 | 2.357576 | 1 | 1.30303 |
| 12 | 404 | 393 | 163 | 210 | 2.478528 | 2.411043 | 1 | 1.288344 |
| 13 | 409 | 390 | 170 | 211 | 2.405882 | 2.294118 | 1 | 1.241176 |
| 14 | 392 | 382 | 159 | 216 | 2.465409 | 2.402516 | 1 | 1.358491 |
| 15 | 410 | 393 | 165 | 219 | 2.484848 | 2.381818 | 1 | 1.327273 |
| 16 | 431 | 394 | 166 | 221 | 2.596386 | 2.373494 | 1 | 1.331325 |
| 17 | 414 | 392 | 166 | 221 | 2.493976 | 2.361446 | 1 | 1.331325 |
| 18 | 444 | 289 | 163 | 211 | 2.723926 | 1.773006 | 1 | 1.294479 |
| 19 | 404 | 292 | 166 | 211 | 2.433735 | 1.759036 | 1 | 1.271084 |
| 20 | 407 | 391 | 164 | 218 | 2.481707 | 2.384146 | 1 | 1.329268 |
| AVG | 406.55 | 380.05 | 165.4 | 214.9 | 2.458014 | | | |
| SD | 12.596 | 30.16 | 3.27 | 7.73 | 3.843576 | | | |
| Norm | 2.4580 | 2.2977 | 1 | 1.294 | 2.45801 | | | |