WRITEUP QUICKSORT

Paul's Notes:

I handled the bulk of the OpenMP implementation. Testing was done locally on an AMD 7600X with DDR5-6000.

Correctness: We made a helper file with functions to verify that sorting worked (quicksort and insertion). This was used during all stages of development to verify that things were working (all thread counts and array sizes). We are assuming that the inputs to the program via argv are valid. Furthermore, as quicksort is standard, I used a source that matches exactly course materials from my Analysis of Algorithms course that was a reference for us to use as a starting place. Citation is included in the comments of the code.

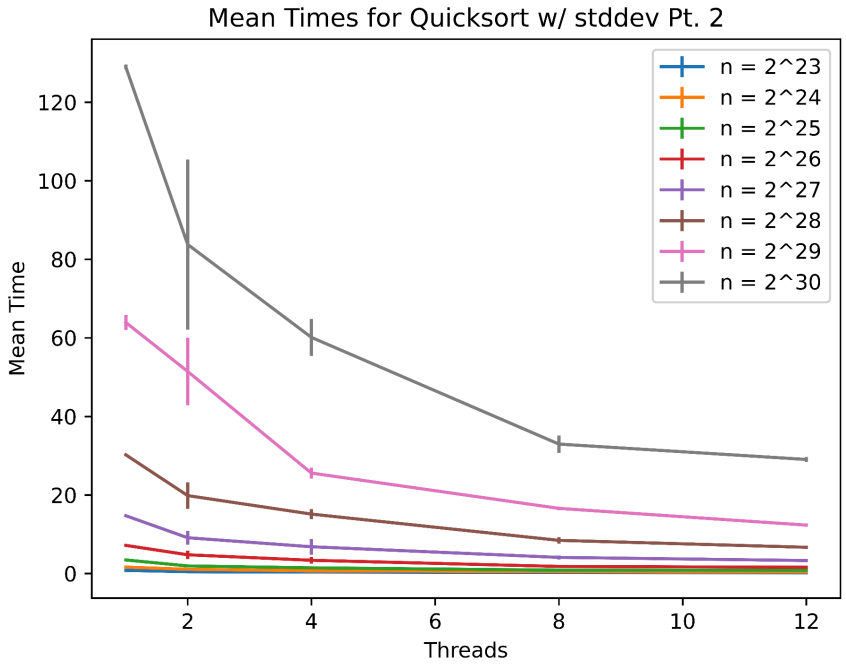
For OpenMP we used a cutoff for subarray sizes (tested 2^n for n in [4,12] - best results were between 64-128 and results are in the cutoff files in results/). Any subarray under this cutoff was sent to insertion sort for processing. The performance improvements were substantial - almost 300% faster for the highest thread count tested (12) at the largest array size (2^30).

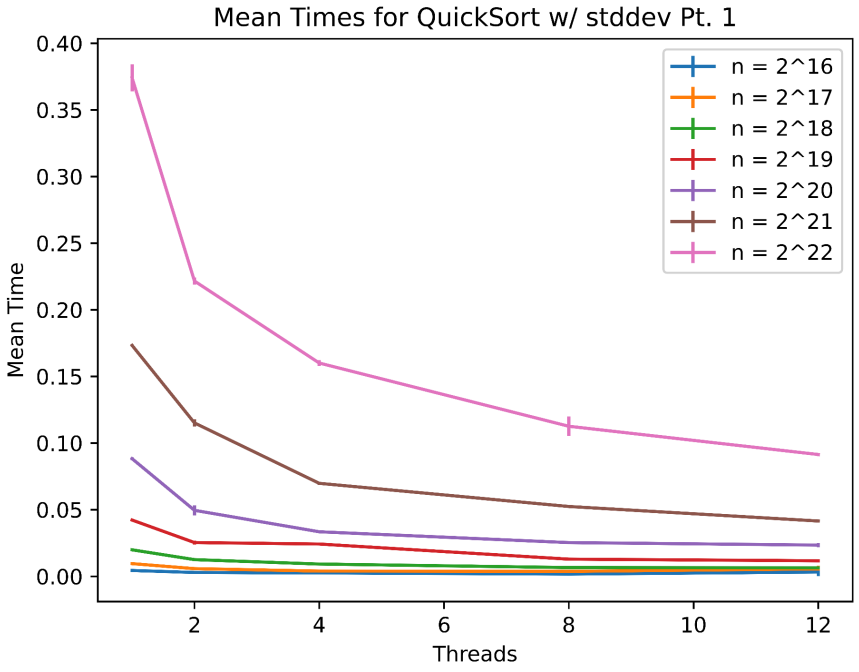
Most development was done with a random pivot selection, which is good for avoiding O(n^2) performance but not as optimal as a median-of-three approach that was implemented for the final results we see here. For larger array sizes, one could implement a more robust sampling method and probably see more gains.

We used Jupyter Notebooks + pandas/numpy/matplotlib to create charts and tables that you see here below. Tables available in those folders as well.

LLMs: Claude proffered the idea of a cutoff of 1000, much smaller values proved to be best. Claude introduced a few off-by-one errors. I generally prefer it for coding over ChatGPT which also had issues or unhelpful suggestions. Once, ChatGPT claimed to fix a bug (when I was looking into what turned out to be an off-by-one issue) and literally gave me the exact same code back. Claude was great for OpenMP and 90% of what it gave me. I still preferred to hand write and they didn't do too much, most of my consultation was asking about implementations of Java and Python etc.'s sorting implementations (dual pivoting, TimSort, etc) which I took at its word.

I don't like relying on LLMs right now, but for some simple tasks it can create a lot of boilerplate fast and I trust myself to debug due to programming fundamentals.


Mean Times for Quicksort w/ stddev Pt. 2
Mean Time
Threads
n = 2^23
n = 2^24
n = 2^25
n = 2^26
n = 2^27
n = 2^28
n = 2^29
n = 2^30
0
20
40
60
80
100
120
2
4
6
8
10
12


Mean Times for QuickSort w/ stddev Pt. 1
Mean Time
Threads
n = 2^16
n = 2^17
n = 2^18
n = 2^19
n = 2^20
n = 2^21
n = 2^22
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
2
4
6
8
10
12

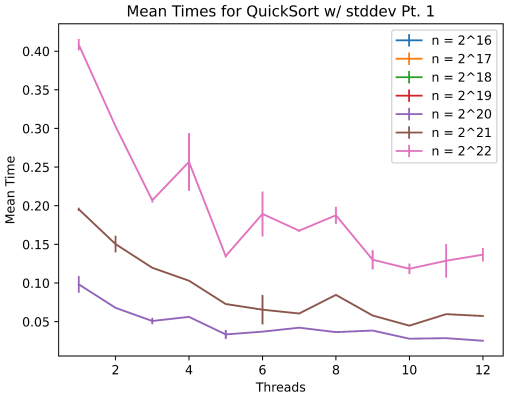
Otto’s Notes(pthread algorithm):

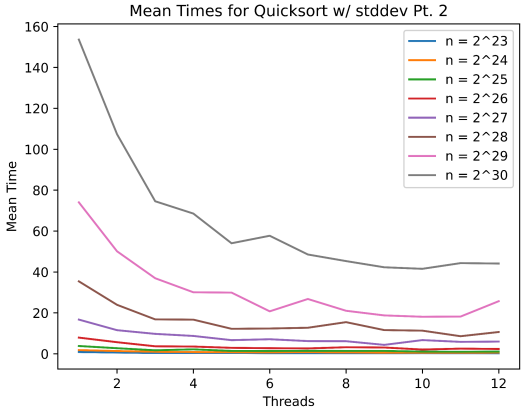
I developed locally on my mac but did my testing on the fox server.

Correctness: I added debugging statements that would ensure that the sorted array is only increasing and tested with random integers to ensure that the sorting algorithm works with different combinations of integers.

Experimentation results: We found the pivot using median of three for our quicksort algorithm which caused a significant speed up for our quick sort algorithm and would switch to insertion sort when our subarray in our thread was less than 32 integers long. I limited the threads by making the max depth of our parallel recursive quicksort to be log_2 of the max number of threads.

Performance Plots for pthreads: (refer to pauls notes for how performance is plotted)





LLM assistance: Originally I was using a global variable called “thread_count” to limit the amount of threads being used but with this method lead to many errors, Chatgpt told me to switch to limiting threads by the depth of recursion.