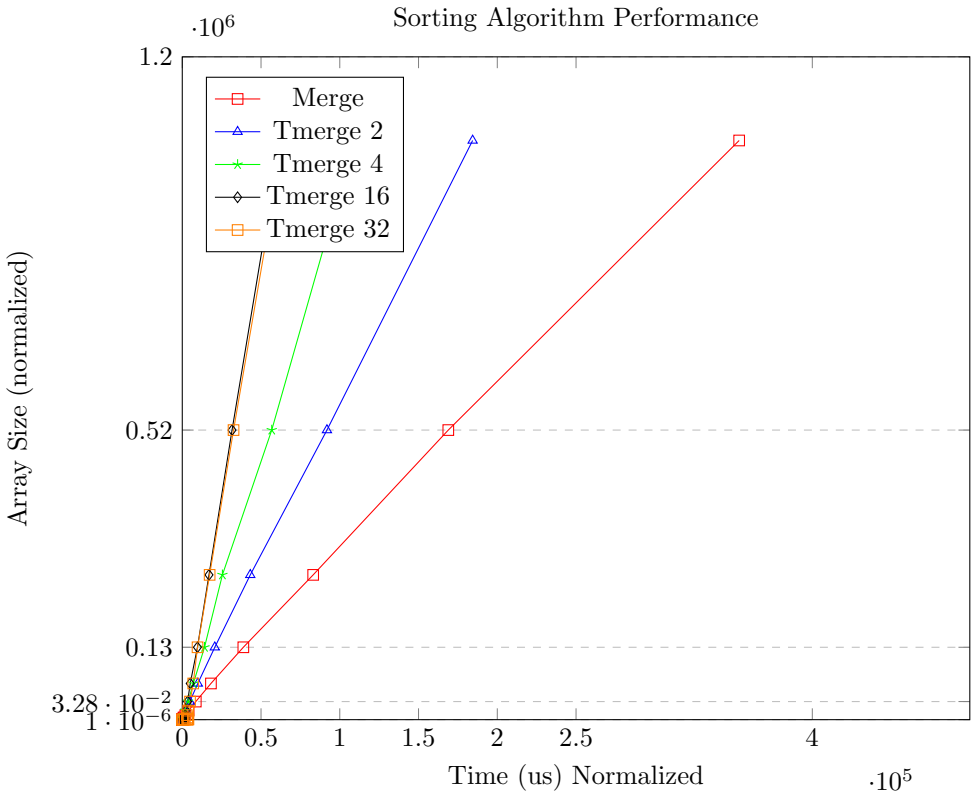
CS361 Homework 4

Nick Pelletier

5/5/2023

1 Table



2 Images and answer to question 2

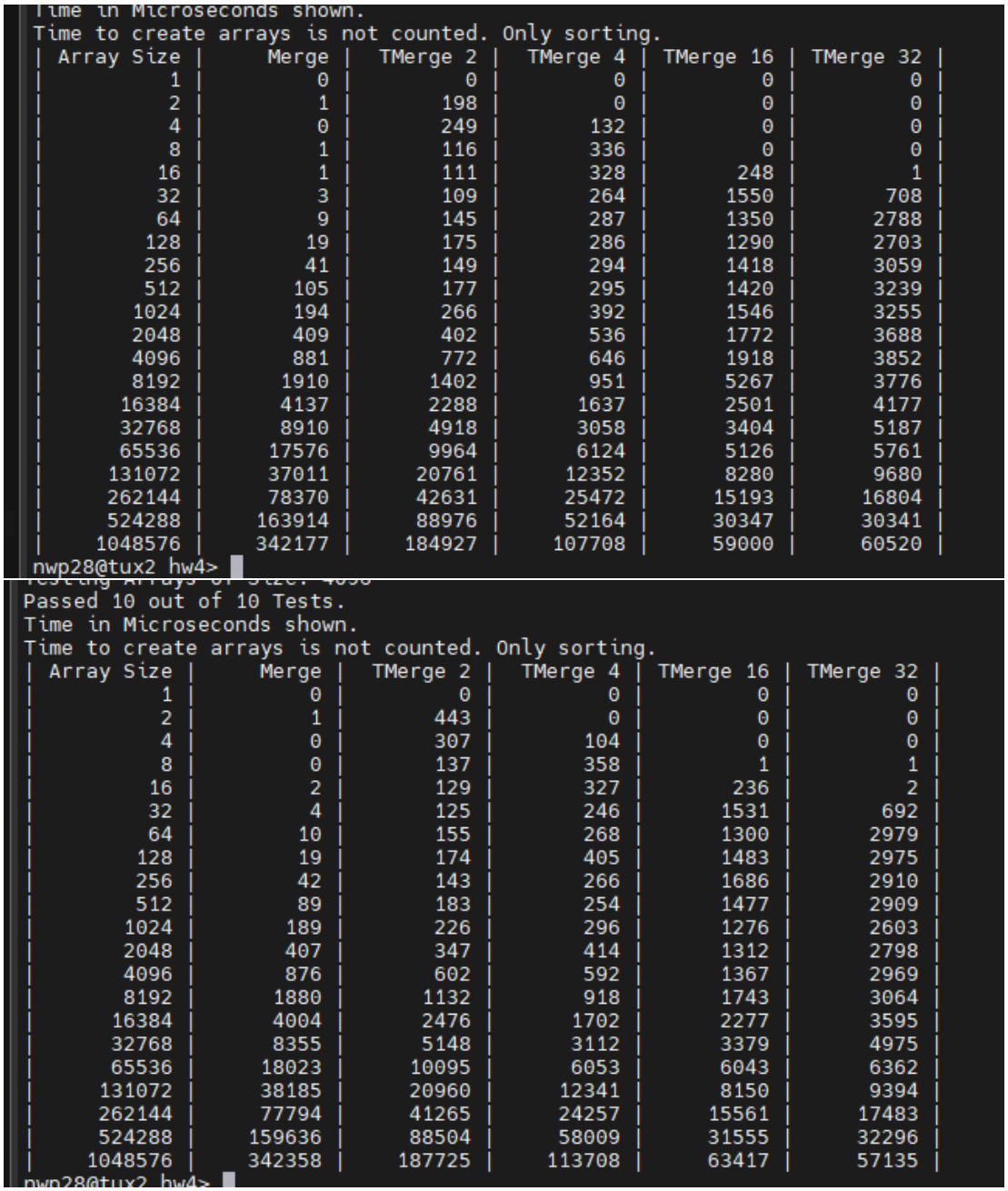
Notice that In the plot we can only see performance at large values. These are the values you'd see at a data center or in general when using large data. As you can see Tmerge32 and Tmerge16 are quite close to each other in my test (this can be partially attributed to the logic I used to determine the correct

number of threads to use).
Below I will show a couple screenshots of table examples from running my code:

```
Passed 10 out of 10 Tests.
Time in Microseconds shown.
Time to create arrays is not counted. Only sorting.
| Array Size |     Merge |  TMerge 2 |  TMerge 4 | TMerge 16 | TMerge 32 |
|          1 |         0 |         0 |         0 |         0 |         0 |
|          2 |        12 |       267 |         0 |         0 |         0 |
|          4 |         0 |        98 |        99 |         0 |         0 |
|          8 |         1 |       180 |       131 |         1 |         0 |
|         16 |         2 |       135 |       378 |       321 |         2 |
|         32 |         4 |       123 |       334 |       875 |       729 |
|         64 |         9 |       139 |       330 |      1764 |      1586 |
|        128 |        20 |       113 |       434 |      1627 |      3364 |
|        256 |        41 |       121 |       273 |      1388 |      3279 |
|        512 |        92 |       209 |       348 |      1624 |      3914 |
|       1024 |       195 |       268 |       352 |      1565 |      3323 |
|       2048 |       439 |       353 |       465 |      1859 |      3653 |
|       4096 |       875 |       589 |       914 |      2006 |      3661 |
|       8192 |      1933 |      1124 |       987 |      1871 |      2897 |
|      16384 |      4036 |      2336 |      1835 |      2282 |      3793 |
|      32768 |      8591 |      4762 |      3703 |      3408 |      4926 |
|      65536 |     18207 |      9858 |      7008 |      5045 |      6950 |
|     131072 |     38769 |     20652 |     14123 |      9630 |      9700 |
|     262144 |     83108 |     43179 |     25491 |     16959 |     17377 |
|     524288 |    168895 |     91850 |     56800 |     31599 |     32552 |
|    1048576 |    353565 |    184351 |    109175 |     62201 |     64191 |
nwp28@tux2 hw4>
```

The above screenshot is the current plot. Seeing this in more detail, we can tell that my method wasn't as good as standard merge sort between the start and 16384 array size. It was only after that (at array size 32768) that my Thread merge sorting method outshone the regular merge sorting method. Of course you can see through these that my usage of threads was quite optimized as the more threads that were given resulted in faster run times.

The next couple screenshots are 2 other results that prove the above statement.



Now where did this improvement come from? When the array size got larger, instead of more recursion with a linear wait-time the thread method split up the recursion into N threads that did their own recursion and resulted in a reduction of time (not completely /n due to laws/algorithms in relation to threading that were explained in the early weeks of class) that can be plainly seen.

3 What was the most difficult portion

Funnily enough, the part that took the most time was the bug fixing and other mistakes like accidentally dividing to create 0 when optimizing thread usage at low array sizes.

4 What was the easiest portion

The easiest portion was compiling my thoughts into something feasible. I took about 1h of thinking on how to make my tmergesort and make it optimal. I got a result I was expecting, but not fully expecting at lower array size values.