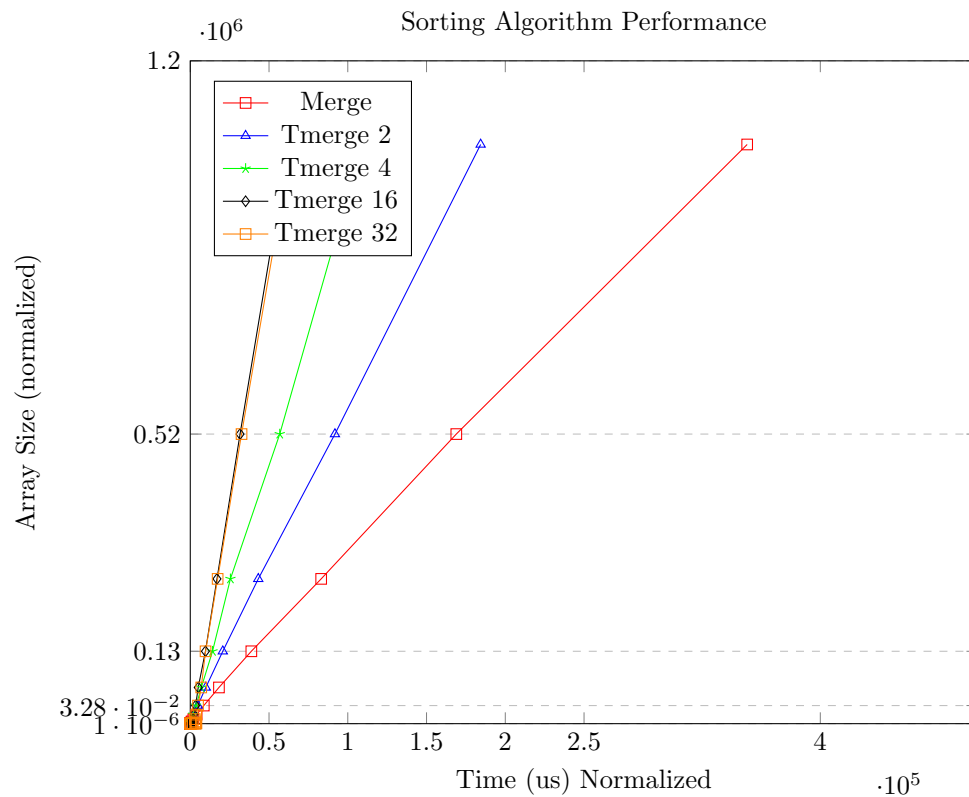


CS361 Homework 4

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.

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	1	198	0	0	0
4	0	249	132	0	0
8	1	116	336	0	0
16	1	111	328	248	1
32	3	109	264	1550	708
64	9	145	287	1350	2788
128	19	175	286	1290	2703
256	41	149	294	1418	3059
512	105	177	295	1420	3239
1024	194	266	392	1546	3255
2048	409	402	536	1772	3688
4096	881	772	646	1918	3852
8192	1910	1402	951	5267	3776
16384	4137	2288	1637	2501	4177
32768	8910	4918	3058	3404	5187
65536	17576	9964	6124	5126	5761
131072	37011	20761	12352	8280	9680
262144	78370	42631	25472	15193	16804
524288	163914	88976	52164	30347	30341
1048576	342177	184927	107708	59000	60520

nwp28@tux2 hw4>

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	1	443	0	0	0
4	0	307	104	0	0
8	0	137	358	1	1
16	2	129	327	236	2
32	4	125	246	1531	692
64	10	155	268	1300	2979
128	19	174	405	1483	2975
256	42	143	266	1686	2910
512	89	183	254	1477	2909
1024	189	226	296	1276	2603
2048	407	347	414	1312	2798
4096	876	602	592	1367	2969
8192	1880	1132	918	1743	3064
16384	4004	2476	1702	2277	3595
32768	8355	5148	3112	3379	4975
65536	18023	10095	6053	6043	6362
131072	38185	20960	12341	8150	9394
262144	77794	41265	24257	15561	17483
524288	159636	88504	58009	31555	32296
1048576	342358	187725	113708	63417	57135

nwp28@tux2 hw4>

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 $1/n$ 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.