Program Structures and Algorithms
Spring 2023(SEC – 3)

NAME: Akshay Parab
NUID: 002766150

Task:
We need to implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. We need to consider two different schemes for deciding whether to sort in parallel:

1. A cut-off. We need to experiment and come up with a good value for this cut-off. If there are fewer elements to sort than the cut-off, then we should use the system sort instead.
2. Recursion depth or the number of available threads. Experiment to decide on an ideal number (t) of separate threads (sticking to the powers of 2) and arrange for that number of partitions to be parallelized (by preventing recursion after the depth of lg (thread) is reached).

Experiments should involve sorting arrays of sufficient size for the parallel sort to make a difference. We need to run with many different array sizes and different cut-off schemes.

Relationship Conclusion:

1. The most optimal efficiency with respect to cut-off value was observed when cut-off value was configured between arraysize/2^4 + 1 & arraysize/2^9+1
2. The most optimal efficiency with respect to thread size was observed when the threads were more than or equal to 8 that is when all the CPU cores (i.e., for my system) were engaged simultaneously
3. There was not much efficiency difference observed between 8 / 16 thread and 32 threads, i.e., 8 to 16 cores were sufficiently optimal for workload distribution. When extra threads are involved than required, CPU time is more as the overhead of managing threads increases (due to the synchronization and communication between threads)
4. When the cut-off size is very large i.e., when the parallelization is turned off early on the process or is never turned on, we observe that irrespective of the thread count we get the same performance, although in most cases such performance is better than deploying more threads than optimally required
5. Parallelization is evidently scalable and possibly the only way to sort ultra-large array in reasonable time, but can be extremely problematic when not switched off during smaller array sizes
6. One downside observed when parallelization of the sort of process was that the memory usage increased more than usual, probably since multiple threads required separate memory buffers simultaneously as opposed to memory usage pattern of sequential sorting algorithms

Evidence to support that conclusion:
Below, you will find data corresponding the following values of array sizes:

1. 1048576
2. 2097152
3. 4194304
4. 8388608
5. 16777216

For one array size, I have used 13 different values of cut-off and for each cut-off value, I have measured time duration in milliseconds by varying the number of threads between 4, 8, 16 and 32, And for each such thread count, I have performed 10 iterations.

As per professor's recommendation, I have chosen cut-off value based on the number of elements such that I can switch from parallel sort to system sort post reaching lg (threads). The number of nodes at level h in a binary tree is given by 2^h. Therefore, I chose cut-off values from arraysize/2^0 + 1 to arraysize/2^{12} + 1, so that I can cut-off parallelization on the node post I reach lg (h) during the sort-process.

My system consists of 8 CPU cores; hence it can run at least 8 threads parallelly.

Also, code changes were required to configure various values of thread count via ForkJoinPool to parallelize the sort process rather than using the default common pool of ForkJoinPool.

Array Size	**Cut-off Value**	**Duration(ms) for 4 Threads**	**Duration(ms) for 8 Threads**	**Duration(ms) for 16 Threads**	**Duration(ms) for 32 Threads**
1048576	1048577	73	72	69	71
1048576	524289	82	42	42	42
1048576	262145	43	29	29	29
1048576	131073	47	38	28	27
1048576	65537	39	41	32	27
1048576	32769	38	37	33	27
1048576	16385	37	32	31	32
1048576	8193	47	31	30	29
1048576	4097	34	31	32	31
1048576	2049	35	33	33	32
1048576	1025	47	47	44	45
1048576	513	91	92	93	101
1048576	257	325	256	274	272

Array Size	**Cut-off Value**	**Duration(ms) for 4 Threads**	**Duration(ms) for 8 Threads**	**Duration(ms) for 16 Threads**	**Duration(ms) for 32 Threads**
2097152	2097153	140	144	135	143
2097152	1048577	86	88	86	85
2097152	524289	89	59	59	59
2097152	262145	98	83	55	55
2097152	131073	84	85	67	56
2097152	65537	68	77	74	56
2097152	32769	72	65	70	62
2097152	16385	63	63	85	60
2097152	8193	62	62	67	62
2097152	4097	66	64	81	71
2097152	2049	72	74	80	85
2097152	1025	115	119	119	121

2097152	513	292	297	283	284

Array Size	Cut-off Value	Duration(ms) for 4 Threads	Duration(ms) for 8 Threads	Duration(ms) for 16 Threads	Duration(ms) for 32 Threads
4194304	4194305	283	294	294	293
4194304	2097153	172	175	176	177
4194304	1048577	177	122	123	137
4194304	524289	205	159	108	112
4194304	262145	183	157	138	123
4194304	131073	174	175	160	121
4194304	65537	134	128	151	132
4194304	32769	127	127	135	134
4194304	16385	130	125	126	141
4194304	8193	140	122	130	136
4194304	4097	132	138	156	137
4194304	2049	168	185	190	172
4194304	1025	367	338	431	360

Array Size	Cutoff Value	Duration(ms) for 4 Threads	Duration(ms) for 8 Threads	Duration(ms) for 16 Threads	Duration(ms) for 32 Threads
8388608	8388609	602	603	591	598
8388608	4194305	356	359	356	353
8388608	2097153	382	252	252	249
8388608	1048577	431	326	226	227
8388608	524289	365	342	281	228
8388608	262145	347	313	292	235
8388608	131073	312	302	277	264
8388608	65537	272	298	275	283
8388608	32769	250	277	276	260
8388608	16385	258	291	272	259
8388608	8193	255	316	297	310
8388608	4097	284	315	338	436
8388608	2049	434	509	473	517

Array Size	Cutoff Value	Duration(ms) for 4 Threads	Duration(ms) for 8 Threads	Duration(ms) for 16 Threads	Duration(ms) for 32 Threads
16777216	16777217	1199	1259	1188	1223
16777216	8388609	727	772	723	739
16777216	4194305	771	579	508	518
16777216	2097153	882	702	441	457
16777216	1048577	808	730	571	462
16777216	524289	593	661	588	481

16777216	262145	597	611	602	556
16777216	131073	531	571	565	553
16777216	65537	538	557	501	536
16777216	32769	550	544	508	507
16777216	16385	593	576	506	556
16777216	8193	542	626	539	649
16777216	4097	661	722	669	814