Program Structures and Algorithms
Spring 2023(SEC 03) Assignment 05

NAME: Shivam Thabe
NUID: 002765286

Task: Parallel Sorting

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

1: A cutoff (defaults to, say, 1000) which you will update according to the first argument in the command line when running. It's your job to experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then you should use the system sort instead.

2: Recursion depth or the number of available threads. Using this determination, you might decide on an ideal number (*t*) of separate threads (stick to powers of 2) and arrange for that number of partitions to be parallelized (by preventing recursion after the depth of *lg t* is reached).

3: An appropriate combination of these.

Implementation:

The array sizes used for the experiment are [1048576, 2097152, 4194304, 8388608] the array size are calculated as the powers of 2, where the power ranges from 20 to 23. Further the thread sizes used are [2, 4, 8, 16, 32]. The cutoff sizes are the array size after which the system sort implementation is used for sorting the array. Randomised data is pushed into the array depending upon the array size above and the average time required to sort is obtained for all the combinations of the threads and cutoff sizes for different arrays.

Console Output Example:

```
size of Array: 1048576 cutoff: 1025 Total threads: 32       Avg time: 190ms
size of Array: 1048576 cutoff: 2049 Total threads: 32       Avg time: 67ms
size of Array: 1048576 cutoff: 4097 Total threads: 32       Avg time: 61ms
size of Array: 1048576 cutoff: 8193 Total threads: 32       Avg time: 62ms
size of Array: 1048576 cutoff: 16385 Total threads: 32      Avg time: 60ms
size of Array: 1048576 cutoff: 32769 Total threads: 32      Avg time: 51ms
size of Array: 1048576 cutoff: 65537 Total threads: 32      Avg time: 59ms
size of Array: 1048576 cutoff: 131073 Total threads: 32     Avg time: 43ms
size of Array: 1048576 cutoff: 262145 Total threads: 32     Avg time: 47ms
size of Array: 1048576 cutoff: 524289 Total threads: 32     Avg time: 63ms
size of Array: 1048576 cutoff: 1048577 Total threads: 32        Avg time: 101ms
size of Array: 2097152 cutoff: 2049 Total threads: 32       Avg time: 121ms
size of Array: 2097152 cutoff: 4097 Total threads: 32       Avg time: 116ms
size of Array: 2097152 cutoff: 8193 Total threads: 32       Avg time: 100ms
size of Array: 2097152 cutoff: 16385 Total threads: 32      Avg time: 104ms
size of Array: 2097152 cutoff: 32769 Total threads: 32      Avg time: 106ms
size of Array: 2097152 cutoff: 65537 Total threads: 32      Avg time: 108ms
size of Array: 2097152 cutoff: 131073 Total threads: 32     Avg time: 86ms
size of Array: 2097152 cutoff: 262145 Total threads: 32     Avg time: 83ms
size of Array: 2097152 cutoff: 524289 Total threads: 32     Avg time: 94ms
size of Array: 2097152 cutoff: 1048577 Total threads: 32        Avg time: 132ms
size of Array: 2097152 cutoff: 2097153 Total threads: 32        Avg time: 211ms
size of Array: 4194304 cutoff: 4097 Total threads: 32       Avg time: 243ms
```

Output Data:

Array Size	Cutoff	Average Time				
		Threads: 2	Threads: 4	Threads: 8	Threads: 16	Threads: 32
1048576	1025	157ms	91ms	131ms	137ms	123ms
1048576	2049	65ms	76ms	75ms	70ms	71ms
1048576	4097	51ms	55ms	60ms	66ms	54ms
1048576	8193	52ms	56ms	70ms	54ms	58ms
1048576	16385	53ms	46ms	51ms	58ms	44ms
1048576	32769	69ms	60ms	56ms	67ms	42ms
1048576	65537	76ms	58ms	63ms	44ms	42ms
1048576	131073	82ms	73ms	56ms	43ms	42ms
1048576	262145	87ms	67ms	45ms	45ms	45ms
1048576	524289	63ms	63ms	63ms	63ms	63ms
1048576	1048577	101ms	102ms	126ms	102ms	102ms

Array Size	Cutoff	Average Time				
		Threads: 2	Threads: 4	Threads: 8	Threads: 16	Threads: 32
2097152	2049	109ms	110ms	113ms	116ms	116ms
2097152	4097	100ms	99ms	100ms	105ms	101ms
2097152	8193	94ms	96ms	167ms	95ms	94ms
2097152	16385	97ms	92ms	100ms	95ms	95ms
2097152	32769	99ms	99ms	105ms	98ms	107ms
2097152	65537	116ms	101ms	102ms	117ms	87ms
2097152	131073	140ms	132ms	122ms	92ms	85ms
2097152	262145	163ms	140ms	110ms	85ms	81ms
2097152	524289	182ms	140ms	93ms	93ms	93ms
2097152	1048577	130ms	131ms	130ms	132ms	131ms
2097152	2097153	213ms	218ms	214ms	216ms	215ms

Array Size	Cutoff	Average Time				
		Threads: 2	Threads: 4	Threads: 8	Threads: 16	Threads: 32
4194304	4097	232ms	241ms	235ms	241ms	220ms
4194304	8193	218ms	227ms	216ms	240ms	224ms
4194304	16385	203ms	201ms	233ms	205ms	197ms
4194304	32769	193ms	198ms	213ms	212ms	225ms
4194304	65537	240ms	223ms	217ms	231ms	207ms
4194304	131073	245ms	230ms	221ms	226ms	176ms
4194304	262145	314ms	308ms	279ms	205ms	172ms
4194304	524289	343ms	323ms	234ms	171ms	170ms
4194304	1048577	394ms	289ms	192ms	197ms	191ms
4194304	2097153	306ms	276ms	273ms	279ms	294ms
4194304	4194305	445ms	444ms	457ms	451ms	449ms

Array Size	Cutoff					
		Threads: 2	Threads: 4	Threads: 8	Threads: 16	Threads: 32
8388608	8193	579ms	473ms	505ms	524ms	480ms
8388608	16385	423ms	410ms	462ms	477ms	431ms
8388608	32769	426ms	410ms	440ms	403ms	440ms
8388608	65537	422ms	427ms	430ms	422ms	445ms
8388608	131073	464ms	496ms	488ms	501ms	424ms
8388608	262145	544ms	469ms	560ms	508ms	357ms
8388608	524289	568ms	615ms	562ms	501ms	398ms
8388608	1048577	649ms	694ms	549ms	363ms	402ms
8388608	2097153	807ms	615ms	397ms	404ms	427ms
8388608	4194305	577ms	574ms	569ms	576ms	574ms
8388608	8388609	923ms	933ms	938ms	937ms	956ms

Conclusion:

1. From the data above, it is observed that the optimal timing for sorting is obtained for the cutoff value ranging from **eq1**: [(array size) / (2 ^ **pow**) + 1] to [(array size) / (2 ^ 15) + 1]. **pow = 12** (for threads: 2,4,8,16) for array size: 1048576
2. It is also observed that as the size of the input array increases, eg. 8388608, the value of **pow** in cutoff range **eq1,** increases.
3. Similarly, for 32 threads, the value of **pow** for optimal sorting ranges from 15 to 18, which increases as the size of the array increases.
4. Also, it was observed that parallel sorting is not optimal for small array sizes.
5. For very large input sizes, parallel sorting algorithms can be much faster than sequential algorithms because they can divide the work among multiple processors, thereby reducing the time required to complete the sort.
6. Overall, the performance benefits of parallel sorting algorithms tend to be most pronounced for very large input sizes, while their overhead can be a disadvantage for small input sizes.