Jaswal, Rajat: NUID-001443168

INFO 6205

PROGRAM STRUCTURES AND ALGORITHMS

FALL 2018

ASSIGNMENT 4

1. **Configuration:**

   My system has two cores and the degree of parallelism is 3.

2. **Conclusion:**

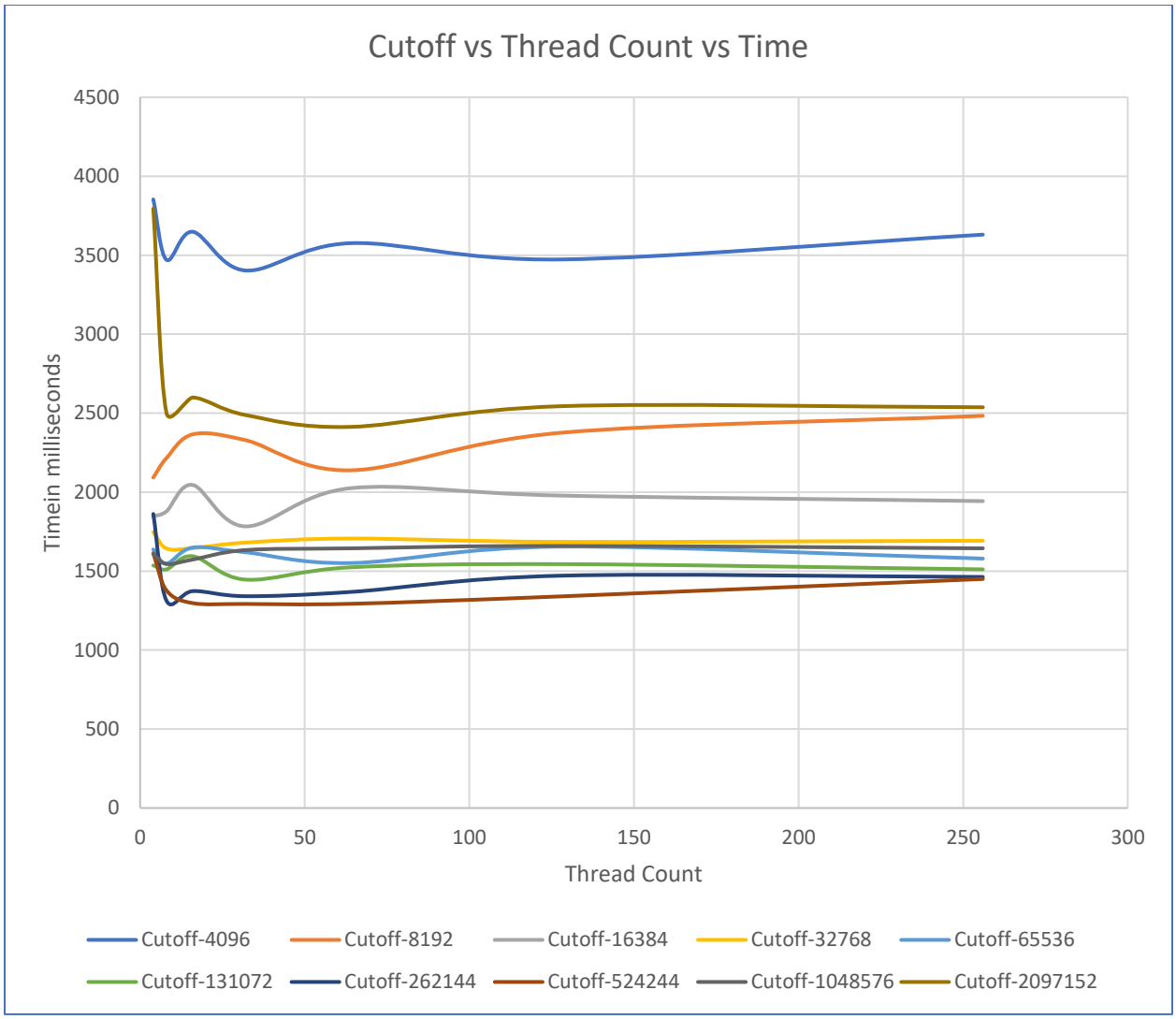
   We are performing the Merge Sort parallelly and comparing the time taken to sort large arrays of size > 2*10^6 as compared to time.

   Since my system has comparatively smaller cores and hence degree of parallelism is comparatively less as compared to other systems.

   The best case of cutoff according to me for T(n)=1292 milliseconds is Cutoff= 52424 and for a thread count of 32.

3. **Graphical Representation:**

Thread Count	Cutoff-4096	Cutoff-8192	Cutoff-16384	Cutoff-32768	Cutoff-65536	Cutoff-131072	Cutoff-262144	Cutoff-524244	Cutoff-1048576	Cutoff-2097152
4	3853	2092	1846	1747	1637	1536	1862	1613	1608	3795
8	3471	2215	1878	1643	1548	1511	1313	1378	1545	2502
16	3648	2366	2045	1645	1649	1594	1373	1297	1573	2599
32	3403	2329	1783	1680	1619	1446	1341	1292	1633	2488
64	3576	2138	2025	1706	1551	1524	1368	1293	1644	2413
128	3473	2376	1978	1685	1655	1544	1471	1340	1660	2544
256	3630	2483	1943	1692	1579	1511	1463	1450	1644	2537



4. PROOF:

By the graph we can depict easily that for thread count of 32 and cutoff- 524244 the time is the minimum. Hence my optimal solution for Parallel sorting depends on two factors

a. The cutoff value.
b. The thread value.

This method is efficient than the normal Arrays.sort() method since this method uses multithreading in which multiple threads perform each operation in parallel and hence highly improves total run time of the sort almost by half.

5. OBSERVATIONS:

Performing parallel sorting on higher arrays and Arrays.sort on smaller array values can be highly efficient.