Abhishikth Daniel Merugu (001548340)

# Program Structures & Algorithms Fall 2021

**Assignment No. 5**

## Task

## 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.

## 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).

## An appropriate combination of these

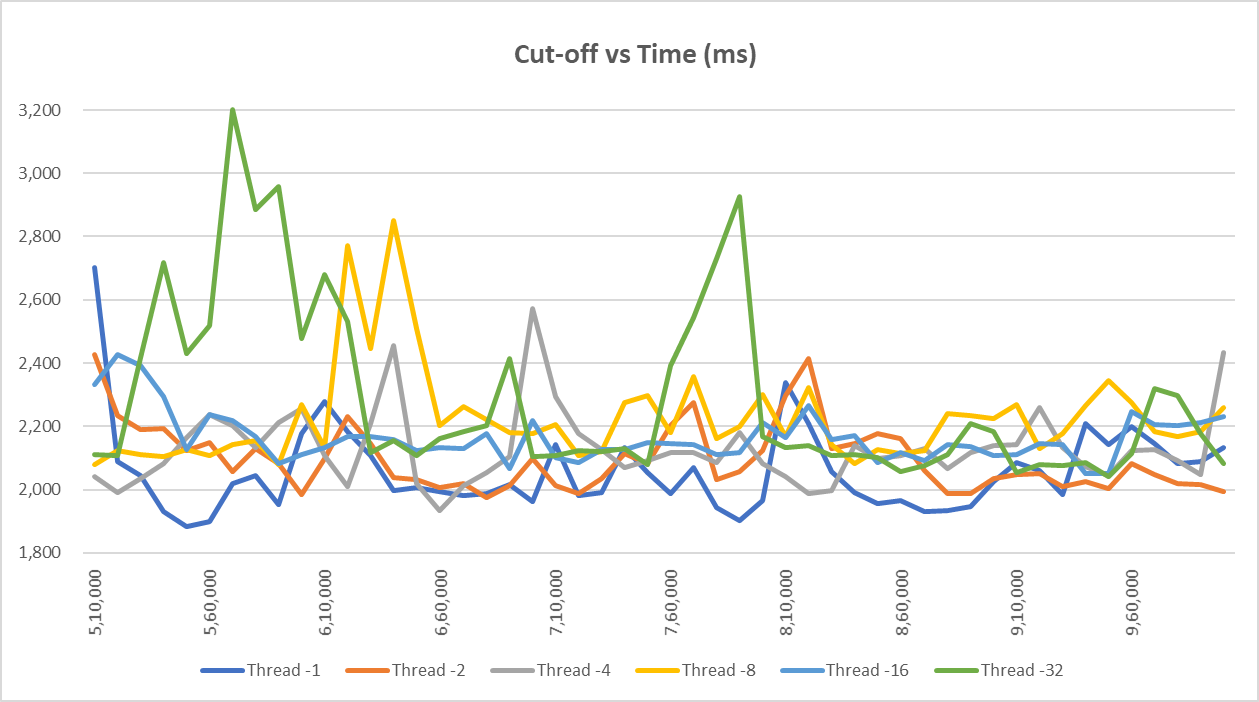
## Implementation:

* + We begin by performing the task for different array sizes. Initial array size is 2 million, by changing the size of the array in Main class line 23.
  + The second task is to perform task for different thread count.
    - We achieve this by externally implement the thread count and passing it to the ForkJoinPool.java executor, which in turn calls the CompletableFuture.java supplyAsync method with the thread count and supply.

## Observations & Analysis:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cut-off** | **Thread -1** | **Thread -2** | **Thread -4** | **Thread -8** | **Thread -16** | **Thread -32** |
| **Average** | 2,058 | 2,097 | 2,129 | 2,226 | 2,165 | 2,287 |
| **Max** | 2,702 | 2,427 | 2,571 | 2,849 | 2,428 | 3,202 |
| **Min** | 1,884 | 1,976 | 1,933 | 2,078 | 2,050 | 2,041 |

### Table.1: Data for different threads and call off for 2 million elements



**Fig.1: Cut-off vs Time for 2 million elements**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cutoff** | **Thread -1** | **Thread -2** | **Thread -4** | **Thread -8** | **Thread -16** | **Thread -32** |
| **Avg** | 4,827 | 4,856 | 4,812 | 4,993 | 5,000 | 4,970 |
| **Max** | 6,720 | 5,375 | 6,445 | 7,298 | 5,816 | 6,903 |
| **Min** | 4,466 | 4,568 | 4,587 | 4,539 | 4,541 | 4,578 |

**Table.2: Data for different threads and call off for 4 million elements**

Thread -4

Thread -32

Thread -2

Thread -16

Thread -1

Thread -8

7,300

6,800

6,300

5,800

5,300

4,800

4,300

Cut-off vs Time (ms)

**Fig.2: Cut-off vs Time for 4 million elements**

5,10,000

5,60,000

6,10,000

6,60,000

7,10,000

7,60,000

8,10,000

8,60,000

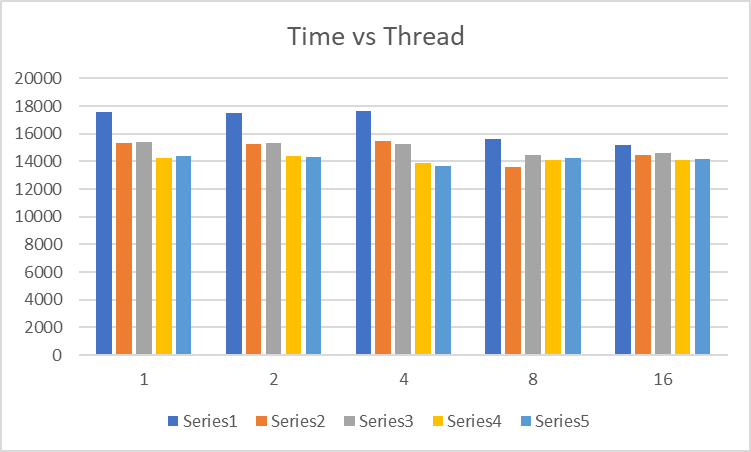
9,10,000

9,60,000

* **Execution time against number of threads:**

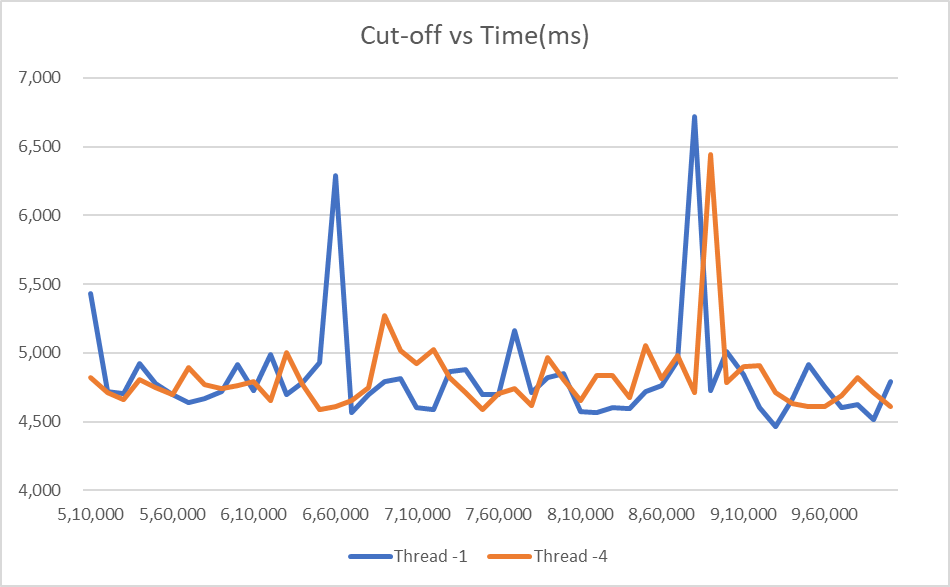
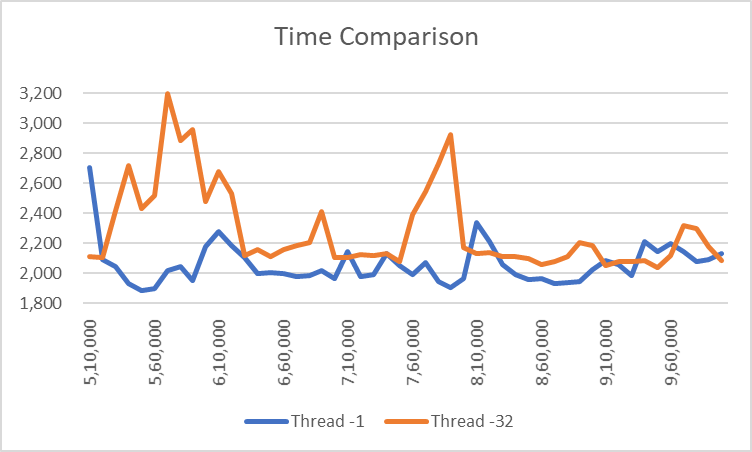
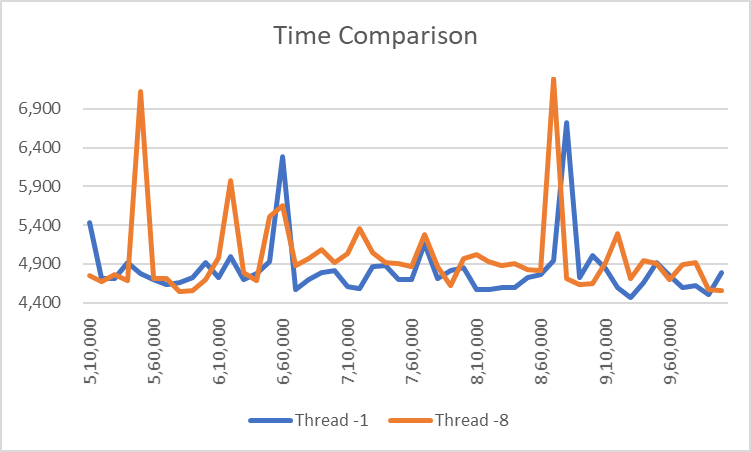
For the experiment, we have kept the cutoff (from 20000 to 100000) and input size of random array fixed (10 million primitive int), the execution time is recorded against the number of threads provided in the pool. The same is plotted using a bar graph (as shown the above screenshot).

It is observed that the execution time to sort 10 million primitive integers over threads 1,2,4 give almost the same performance whereas it falls when the thread pool is increased to 8. However, it shows no significant gain when the thread pool is further increased to hold 16 threads.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of threads** | **Time in ms** | | | | |
| **1** | 17599 | 15329 | 15400 | 14282 | 14403 |
| **2** | 17495 | 15287 | 15353 | 14400 | 14332 |
| **4** | 17671 | 15465 | 15288 | 13865 | 13665 |
| **8** | 15647 | 13608 | 14453 | 14100 | 14237 |
| **16** | 15220 | 14491 | 14641 | 14087 | 14170 |
| **Cutoff** | **20000** | **40000** | **60000** | **80000** | **100000** |

## Conclusion:



* We can infer that the quickest cut-off range is from 650,000 to 750,000. Considering the experimental errors.
* When the cut-off size is too small, single-threaded sort is the in multithreaded sorting algorithms, the overhead of managing so many partitions becomes too large.

### After 4 threads, we have diminishing returns and benefits.

* **Note:** The above results are for my system and may provide different values for other systems. But the conclusion will remain same.