

# Program Structures and Algorithms

## Spring 2024

**NAME:** Aneesh Arunjunai Saravanan

**NUID:** 002675639

**GITHUB LINK:** <https://github.com/aneesharunjunai/INFO6205>

### **Task: Assignment 5 – Parallel Sorting**

Your 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.

There is a Main class and the ParSort class in the sort.par package of the INFO6205 repository. The Main class can be used as is, but the ParSort class needs to be implemented where you see "TODO..." [it turns out that these TODOs are already implemented].

Unless you have a good reason not to, you should just go along with the Java8-style future implementations provided for you in the class repository.

You must prepare a report that shows the results of your experiments and draws a conclusion (or more) about the efficacy of this method of parallelizing sort. Your experiments should involve sorting arrays of sufficient size for the parallel sort to make a difference. You should run with many different array sizes (they must be sufficiently large to make parallel sorting worthwhile, obviously) and different cutoff schemes.

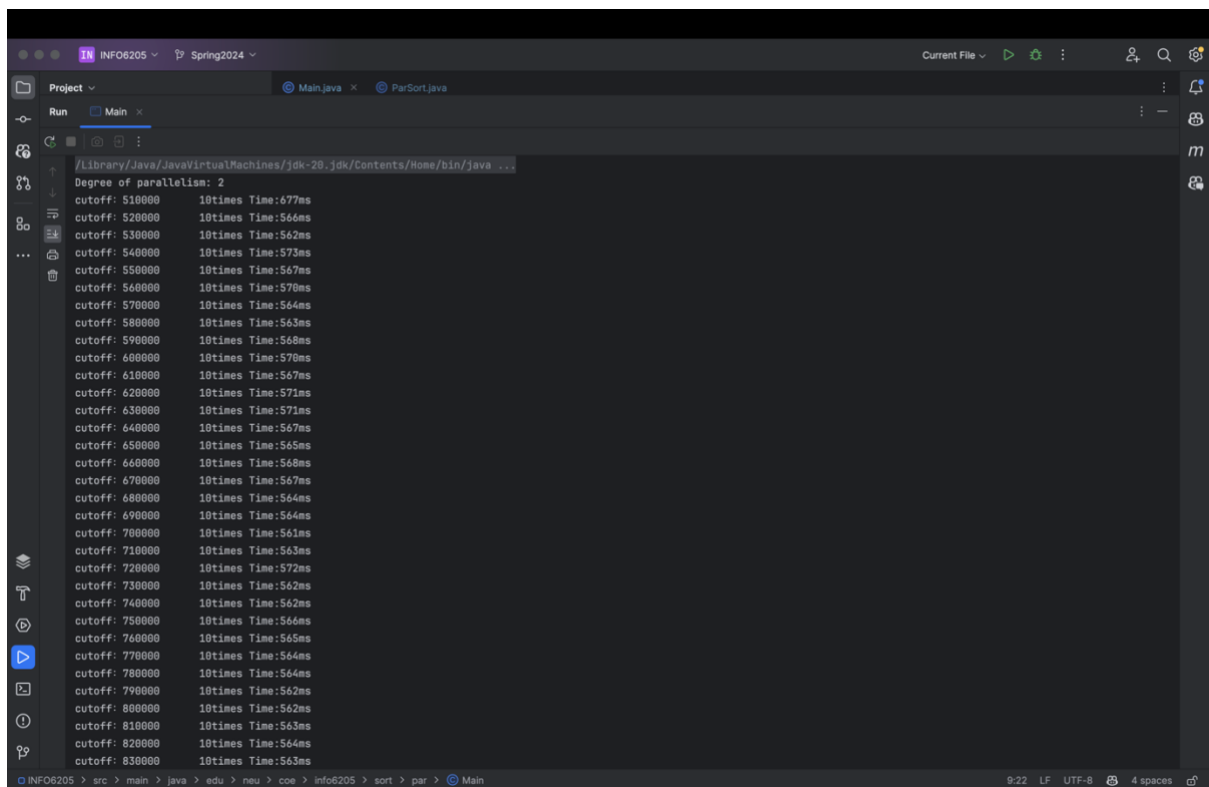
## Observation & Conclusion:

- The optimal cutoff value seems to be around 50,000 across all degrees of parallelism.
- Increasing the cutoff value beyond 50,000 generally leads to longer sorting times.
- Higher degrees of parallelism (11 DOP) can improve performance compared to lower DOP (2 DOP) at lower cutoff values (around 20,000).
- However, the benefit of higher DOP diminishes or even reverses at larger cutoff values (above 50,000).
- This suggests that the overhead of managing more threads outweighs the benefit of parallel processing for larger subtasks.

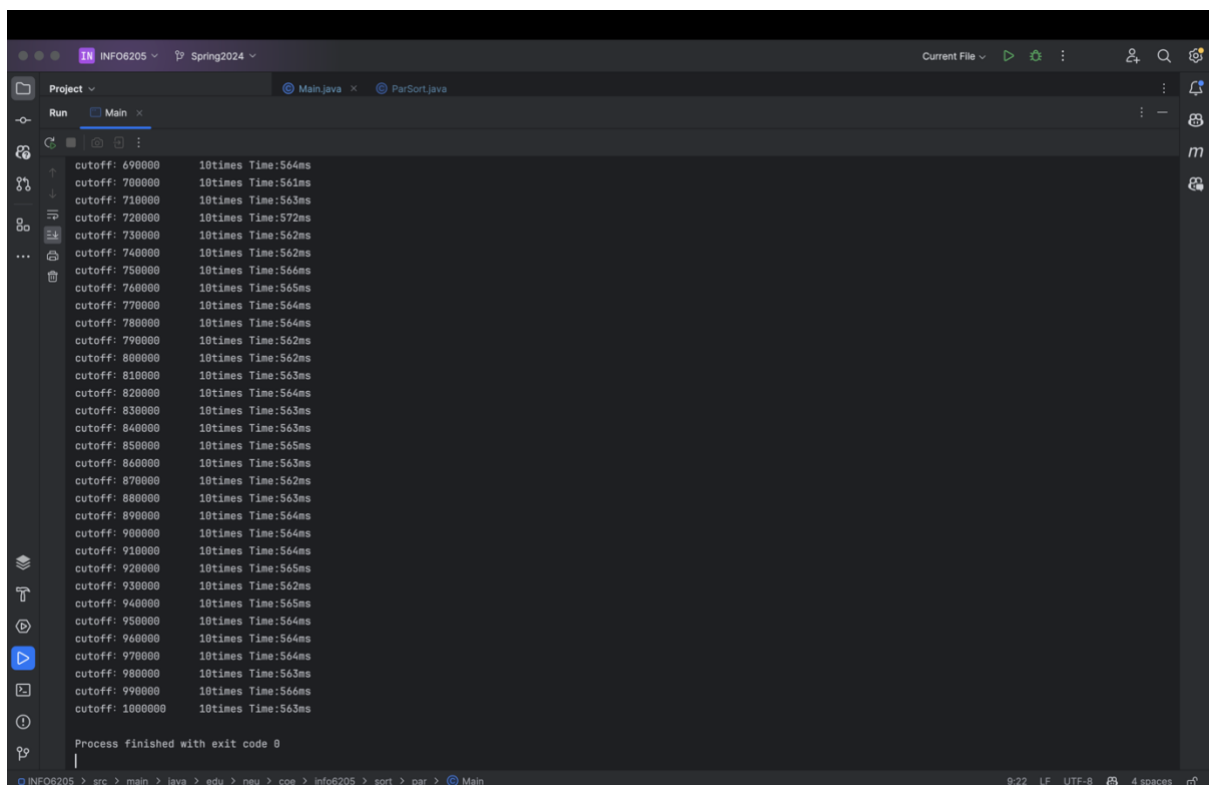
Overall, it appears that there's a trade-off between the cutoff value and the degree of parallelism. A cutoff of around 50,000 seems to be a good starting point, and the optimal DOP depends on the specific hardware and problem size.

With a smaller data set, the overhead of creating and managing parallel tasks might outweigh the benefits of parallelism, making a sequential sort potentially faster. Conversely, with a much larger data set, the benefits of parallelization could become more pronounced. Further experimentation with different array sizes would be necessary to establish a more comprehensive relationship between dataset size, optimal cutoff value, and the ideal degree of parallelism.

# Console Output:



```
INFO6205 Spring2024
Project: Main.java, ParSort.java
Run: Main
/Library/Java/JavaVirtualMachines/jdk-20_jdk/Contents/Home/bin/java ...
Degree of parallelism: 2
cutoff: 510000 10times Time:677ms
cutoff: 520000 10times Time:566ms
cutoff: 530000 10times Time:562ms
cutoff: 540000 10times Time:573ms
cutoff: 550000 10times Time:567ms
cutoff: 560000 10times Time:570ms
cutoff: 570000 10times Time:564ms
cutoff: 580000 10times Time:563ms
cutoff: 590000 10times Time:568ms
cutoff: 600000 10times Time:570ms
cutoff: 610000 10times Time:567ms
cutoff: 620000 10times Time:571ms
cutoff: 630000 10times Time:571ms
cutoff: 640000 10times Time:567ms
cutoff: 650000 10times Time:565ms
cutoff: 660000 10times Time:568ms
cutoff: 670000 10times Time:567ms
cutoff: 680000 10times Time:564ms
cutoff: 690000 10times Time:564ms
cutoff: 700000 10times Time:561ms
cutoff: 710000 10times Time:563ms
cutoff: 720000 10times Time:572ms
cutoff: 730000 10times Time:562ms
cutoff: 740000 10times Time:562ms
cutoff: 750000 10times Time:566ms
cutoff: 760000 10times Time:565ms
cutoff: 770000 10times Time:564ms
cutoff: 780000 10times Time:564ms
cutoff: 790000 10times Time:562ms
cutoff: 800000 10times Time:562ms
cutoff: 810000 10times Time:563ms
cutoff: 820000 10times Time:564ms
cutoff: 830000 10times Time:563ms
```



```
INFO6205 Spring2024
Project: Main.java, ParSort.java
Run: Main
cutoff: 690000 10times Time:564ms
cutoff: 700000 10times Time:561ms
cutoff: 710000 10times Time:563ms
cutoff: 720000 10times Time:572ms
cutoff: 730000 10times Time:562ms
cutoff: 740000 10times Time:562ms
cutoff: 750000 10times Time:566ms
cutoff: 760000 10times Time:565ms
cutoff: 770000 10times Time:564ms
cutoff: 780000 10times Time:564ms
cutoff: 790000 10times Time:562ms
cutoff: 800000 10times Time:562ms
cutoff: 810000 10times Time:563ms
cutoff: 820000 10times Time:564ms
cutoff: 830000 10times Time:563ms
cutoff: 840000 10times Time:563ms
cutoff: 850000 10times Time:565ms
cutoff: 860000 10times Time:563ms
cutoff: 870000 10times Time:562ms
cutoff: 880000 10times Time:563ms
cutoff: 890000 10times Time:564ms
cutoff: 900000 10times Time:564ms
cutoff: 910000 10times Time:564ms
cutoff: 920000 10times Time:565ms
cutoff: 930000 10times Time:562ms
cutoff: 940000 10times Time:565ms
cutoff: 950000 10times Time:564ms
cutoff: 960000 10times Time:564ms
cutoff: 970000 10times Time:564ms
cutoff: 980000 10times Time:563ms
cutoff: 990000 10times Time:566ms
cutoff: 1000000 10times Time:563ms
Process finished with exit code 0
```

## ParSort Graph:

