

Program Structures & Algorithms
Spring 2022
Assignment No. 4 (Parallel Sorting)

Name: Prathamesh Nemade

NUID: 002139730

Task:

Please see the presentation on *Assignment on Parallel Sorting* under the *Exams. etc.* module.

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. For varying the number of threads available, you might want to consult the following resources:

- <https://www.callicoder.com/java-8-completablefuture-tutorial/#a-note-about-executor-and-thread-pool> (Links to an external site.)
- <https://stackoverflow.com/questions/36569775/how-to-set-forkjoinpool-with-the-desired-number-of-worker-threads-in-completable> (Links to an external site.)

Good luck and enjoy.

Output:

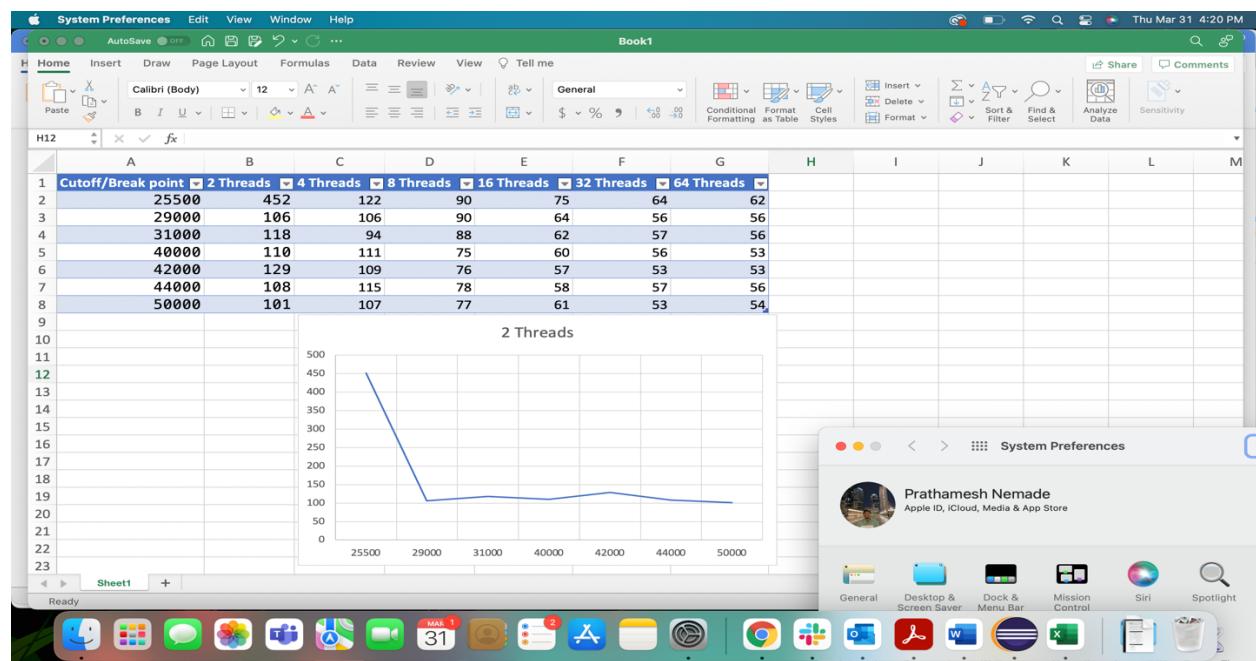
Threads: 2

Cutoff Range: 25500 – 50000

Array Size: 200000

The screenshot shows a Mac desktop with the Dock at the bottom. An Eclipse IDE window is open, displaying code for a Java application named Main.java. The code includes imports for various graph classes and a main method. Below the code, the terminal window shows the execution of the program with an array size of 200000. The output lists times for different cutoff values from 25500 to 36000, showing a general downward trend in execution time as the cutoff value increases. To the right of the Eclipse window, the System Preferences window is also visible.

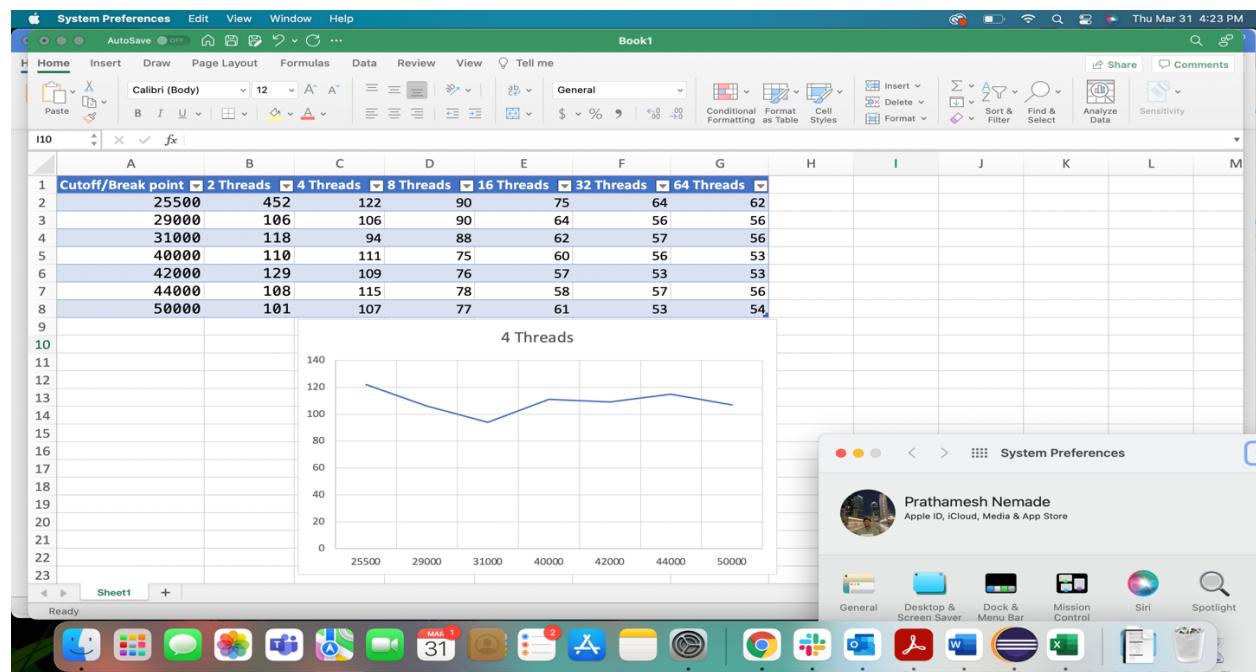
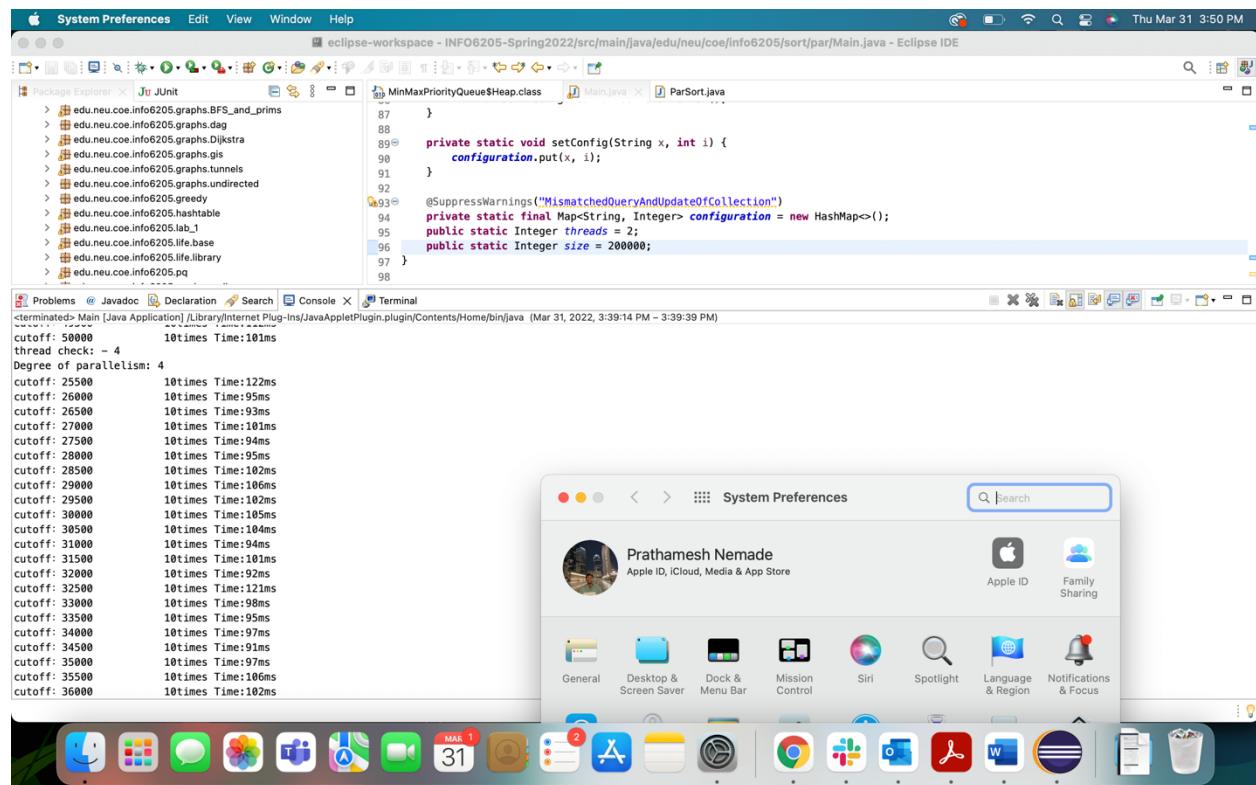
```
Array Size: 200000
thread check: - 2
Degree of parallelism: 2
cutoff: 25500 10times Time:452ms
cutoff: 26000 10times Time:139ms
cutoff: 26500 10times Time:127ms
cutoff: 27000 10times Time:123ms
cutoff: 27500 10times Time:114ms
cutoff: 28000 10times Time:114ms
cutoff: 28500 10times Time:127ms
cutoff: 29000 10times Time:106ms
cutoff: 29500 10times Time:109ms
cutoff: 30000 10times Time:122ms
cutoff: 30500 10times Time:109ms
cutoff: 31000 10times Time:118ms
cutoff: 31500 10times Time:142ms
cutoff: 32000 10times Time:112ms
cutoff: 32500 10times Time:113ms
cutoff: 33000 10times Time:117ms
cutoff: 33500 10times Time:112ms
cutoff: 34000 10times Time:114ms
cutoff: 34500 10times Time:109ms
cutoff: 35000 10times Time:111ms
cutoff: 35500 10times Time:108ms
cutoff: 36000 10times Time:109ms
```



Threads: 4

Cutoff Range: 25500 – 50000

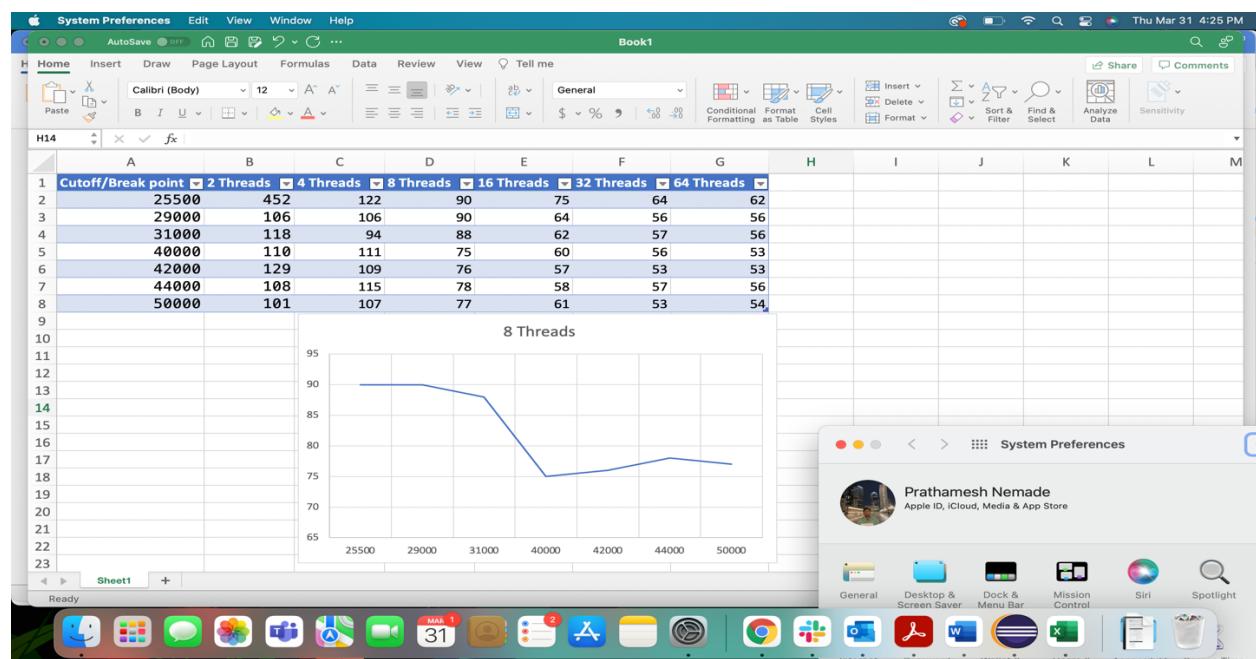
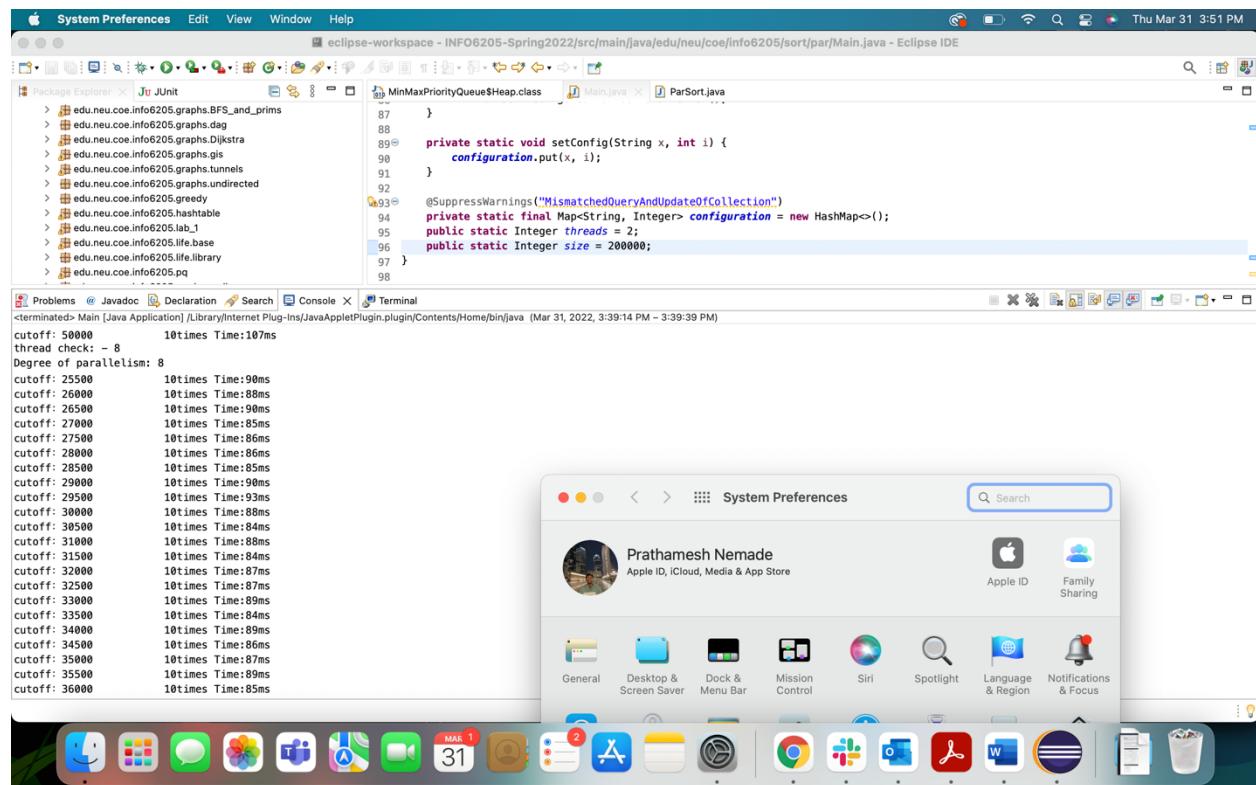
Array Size: 200000



Threads: 8

Cutoff Range: 25500 – 50000

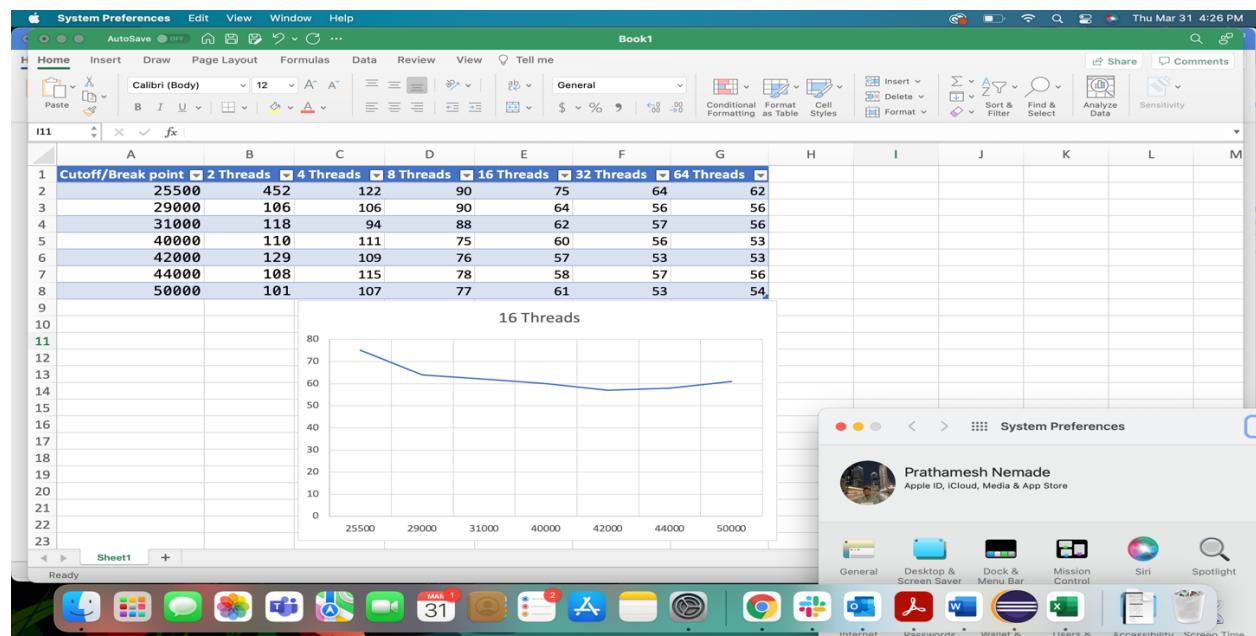
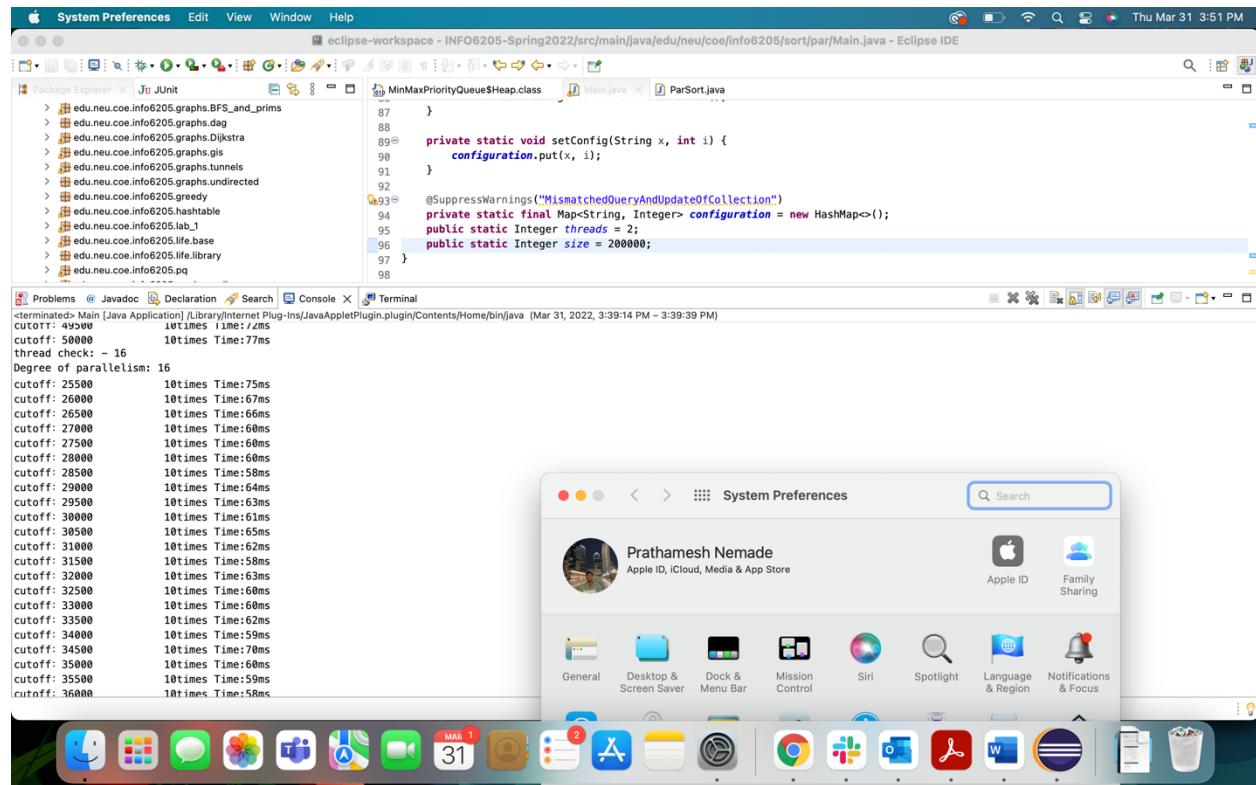
Array Size: 200000



Threads: 16

Cutoff Range: 25500 – 50000

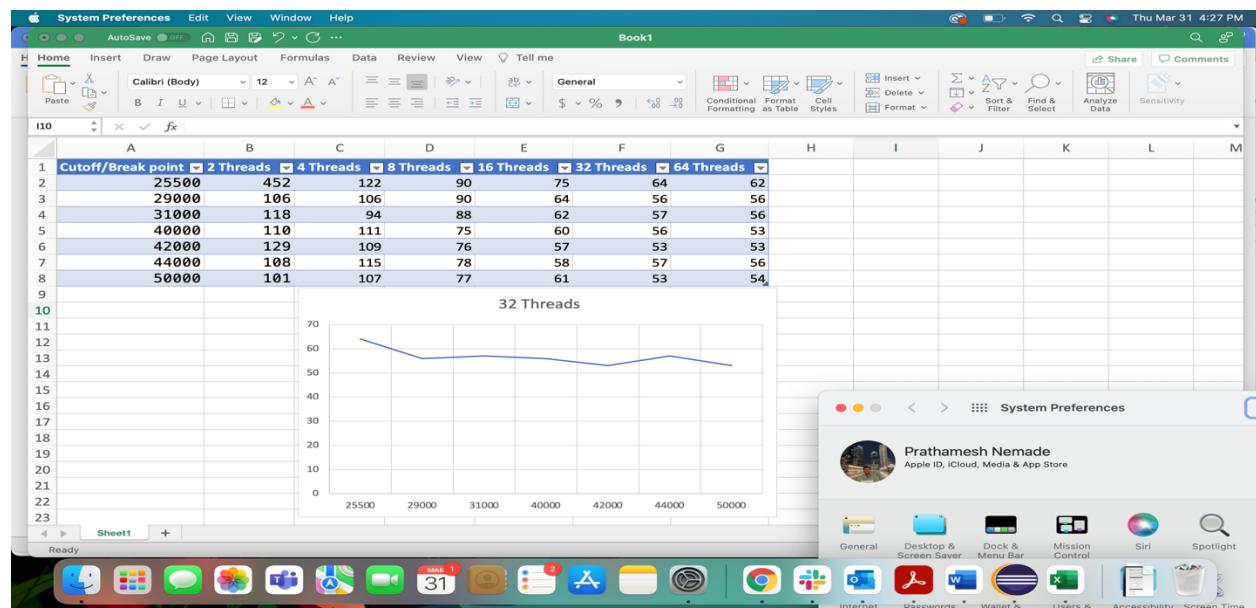
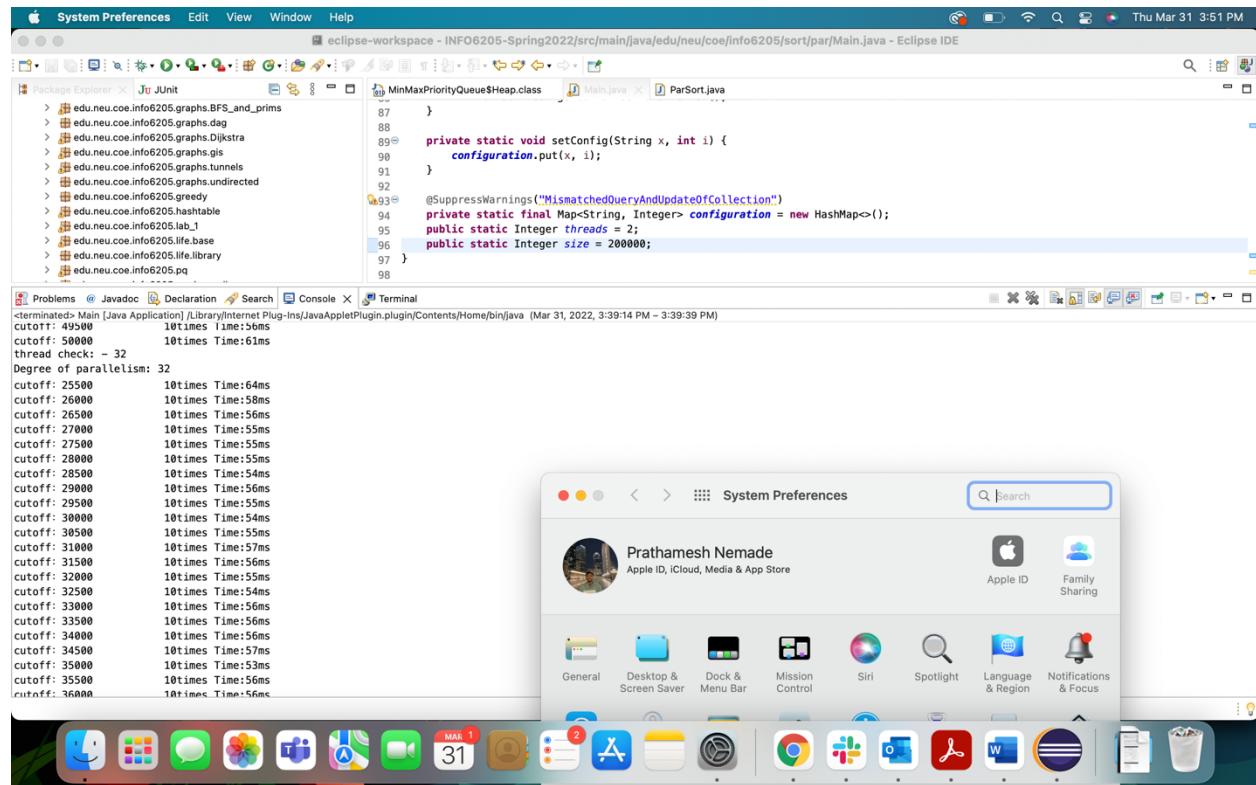
Array Size: 200000



Threads: 32

Cutoff Range: 25500 – 50000

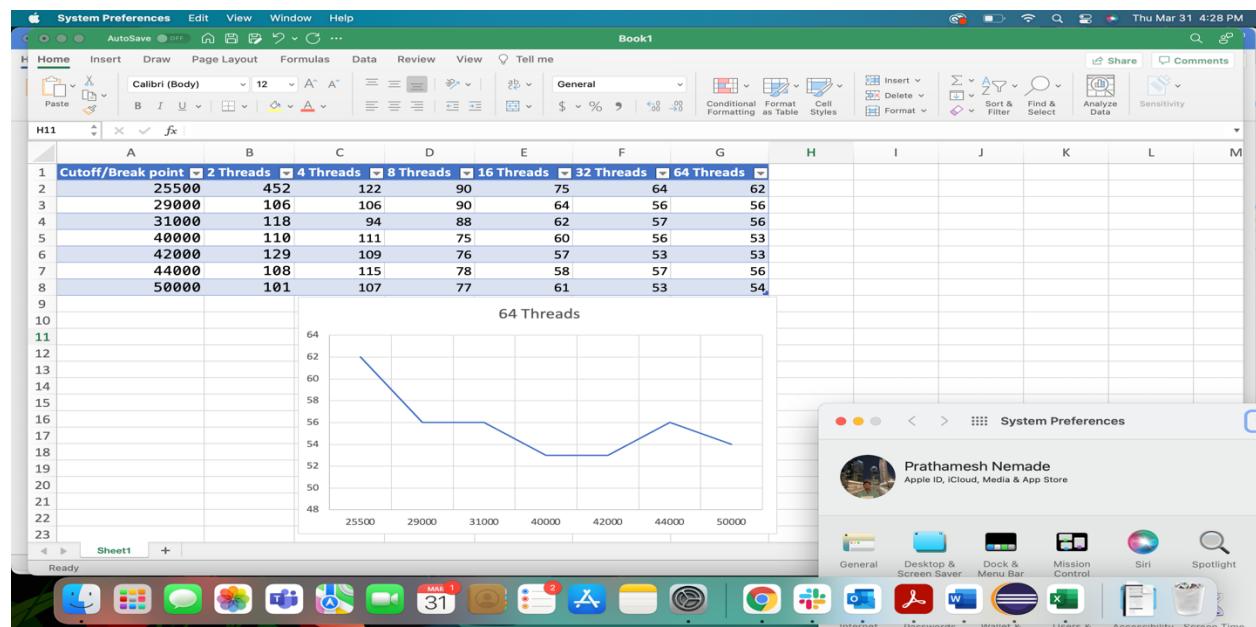
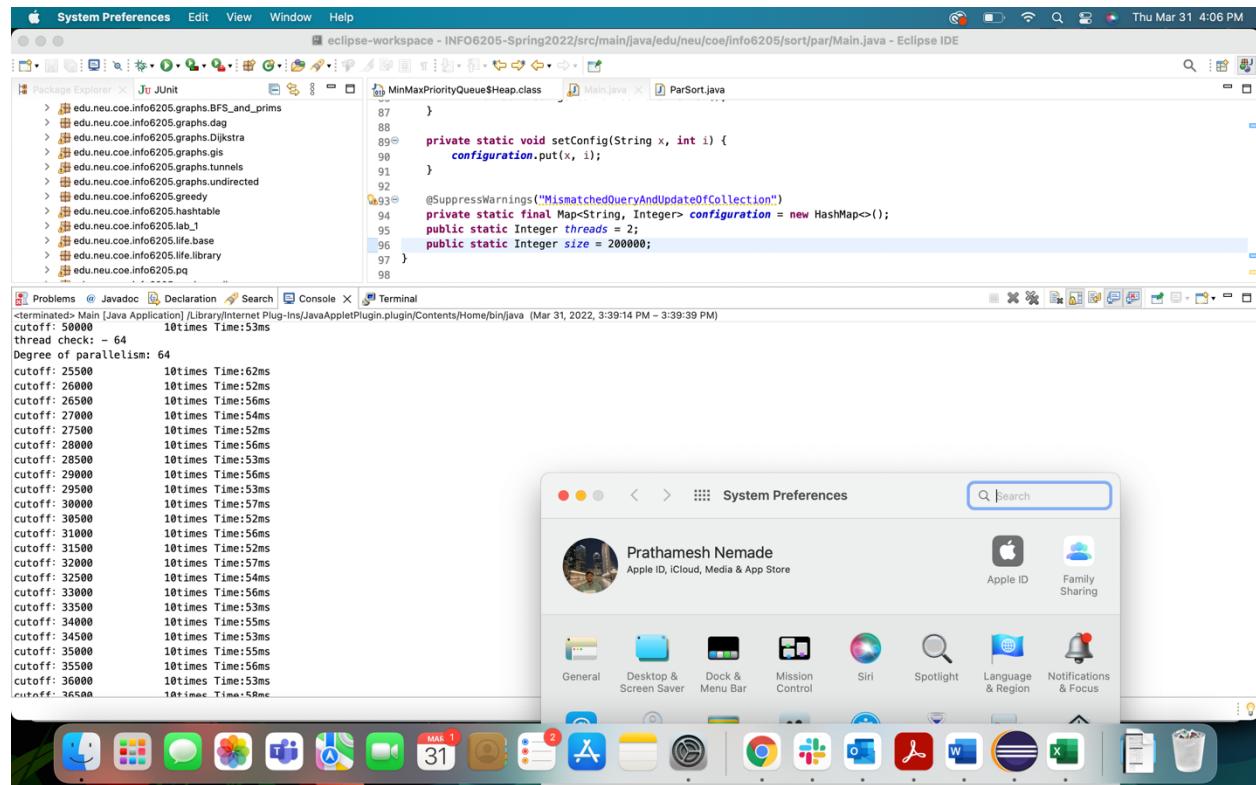
Array Size: 200000



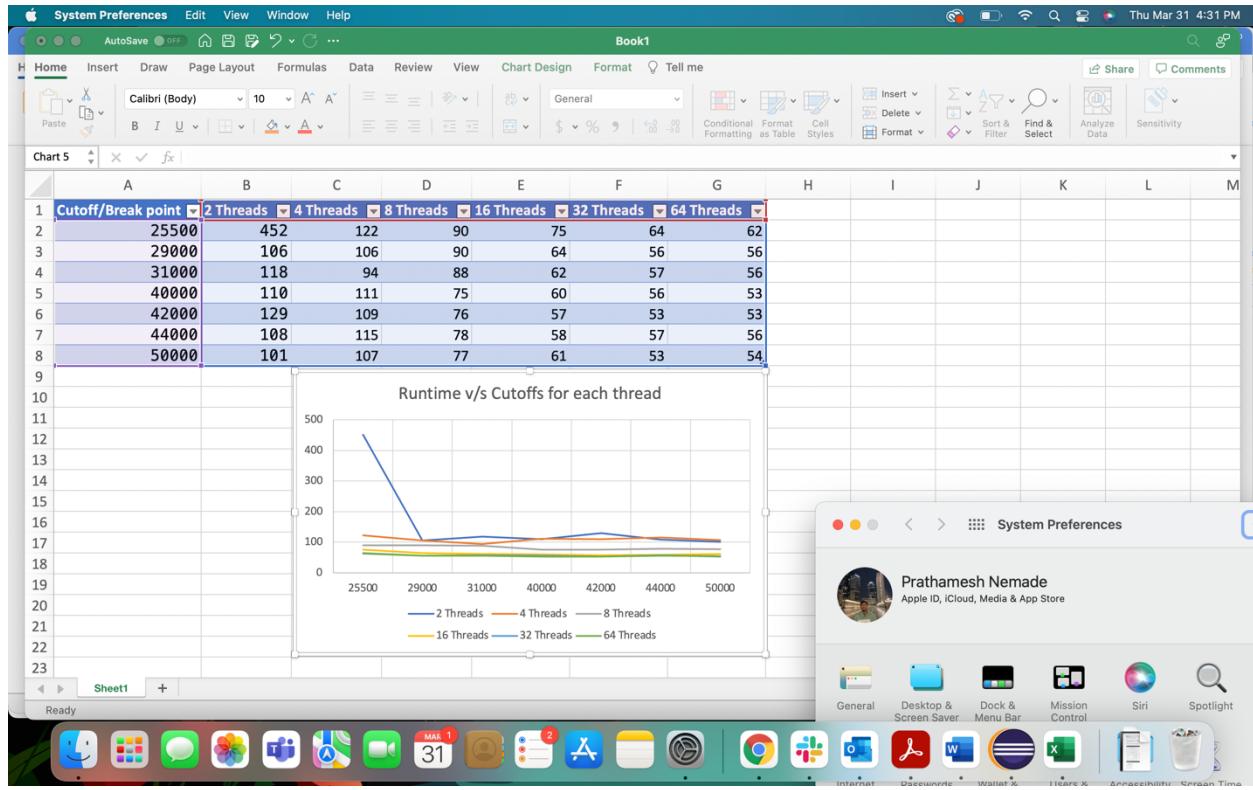
Threads: 64

Cutoff Range: 25500 – 50000

Array Size: 200000



Conclusion:



Running the given algorithm with different values of threads (2 to 64) and the array sizes, the performance is pretty stable after 4/ ~8 threads. The thread is changed at every step by the factor of 2. Therefore, total no of threads used at every step is 2^d , where d is the recursion depth.

Therefore, based on the data we can say, **maximum depth = $\log(\text{arraySize}/\text{cutoffValue})$**