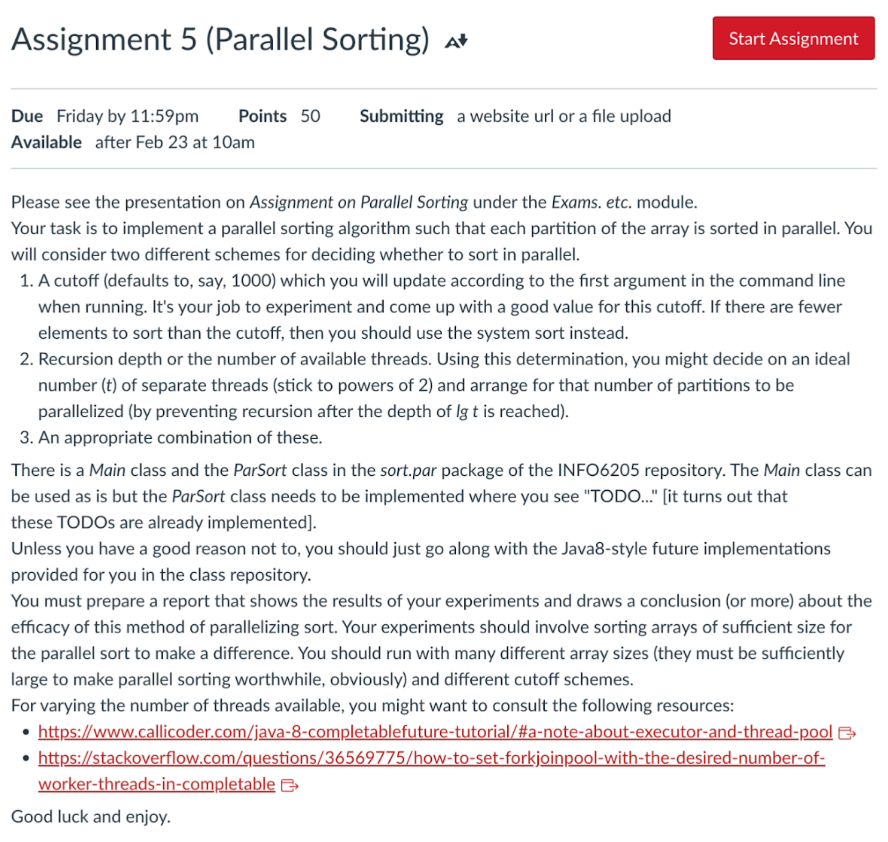
PROGRAM STRUCTURES AND ALGORITHMS

NAME - RAHUL POTHIRENDI
NUID - 002889957
Github - https://github.com/Pothirendirahul/INFO6205.git

TASK



Assignment 5 (Parallel Sorting)

Start Assignment

Due Friday by 11:59pm **Points** 50 **Submitting** a website url or a file upload
Available after Feb 23 at 10am

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.

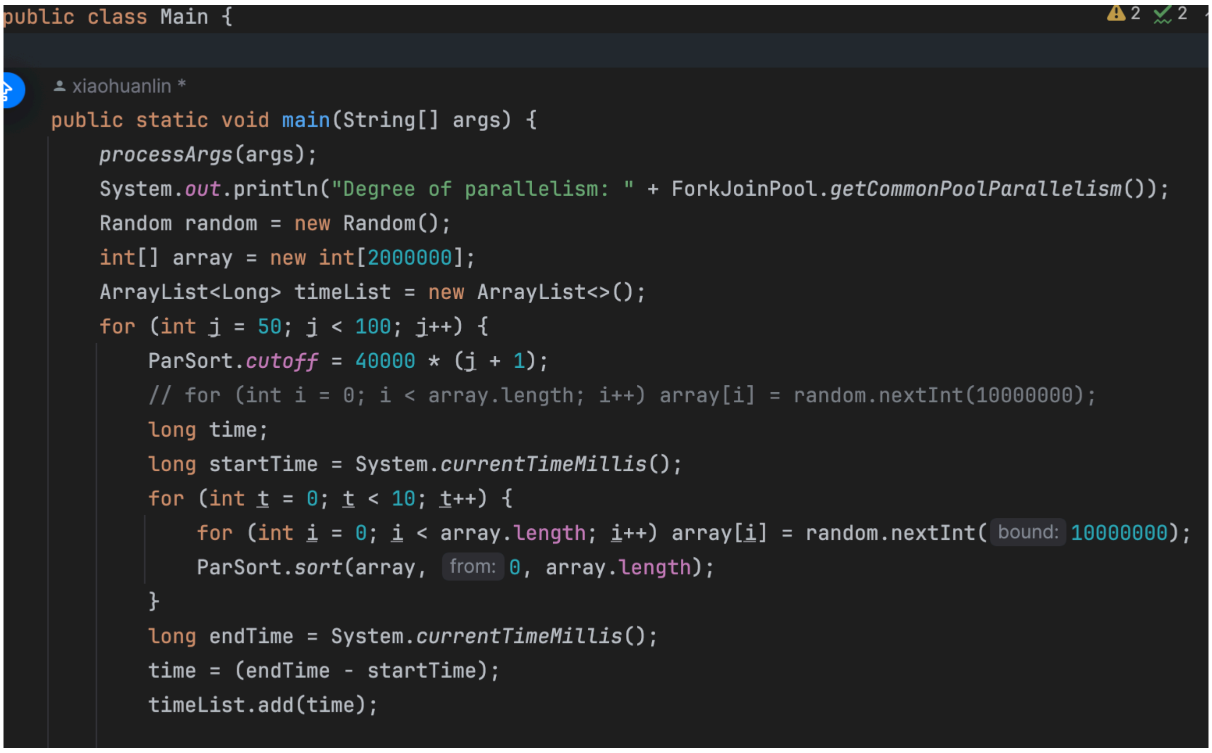
There is a *Main* class and the *ParSort* class in the *sort.par* package of the INFO6205 repository. The *Main* class can be used as is but the *ParSort* class needs to be implemented where you see "TODO..." [it turns out that these TODOs are already implemented].
Unless you have a good reason not to, 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
- https://stackoverflow.com/questions/36569775/how-to-set-forkjoinpool-with-the-desired-number-of-worker-threads-in-completable

Good luck and enjoy.

The code states that Array.sort() will be used to sort the array when its length is less than cutoff. In the event that the array's length exceeds the cutoff, our own sort—a form of merge sort—would be used by the algorithm. Dual-Pivot Quicksort is now the foundation for Array.sort().

Due to its decreased overhead and effective data partitioning approach, Dual-Pivot Quicksort may be quicker than parallel merge sort for small to medium-sized datasets or on single-core machines. Because parallel merge sort may fully exploit parallel processing capabilities to speed up the sorting process, it may provide higher performance for huge datasets, especially on multi-core or multiprocessor computers.

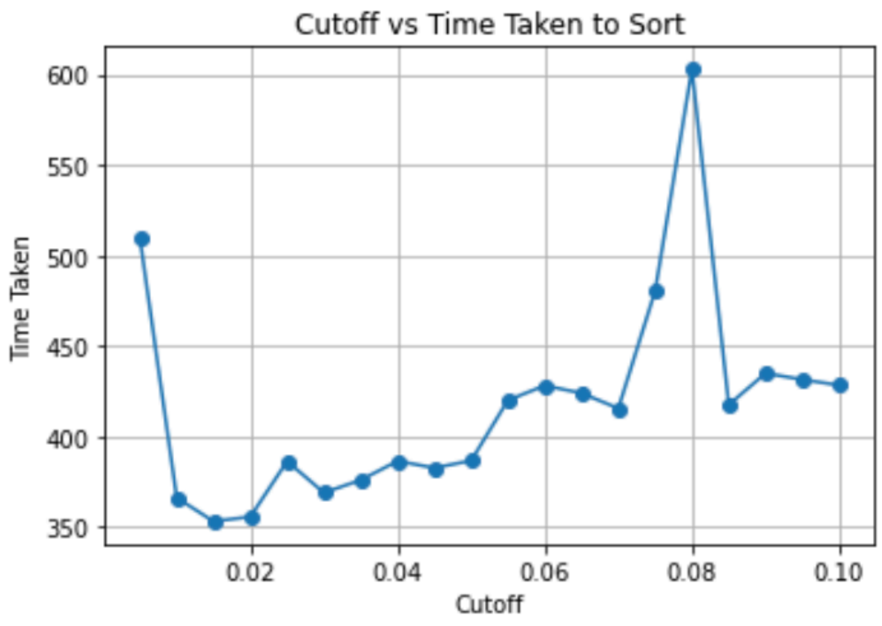


```
public class Main {

    xiaohuanlin *
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + ForkJoinPool.getCommonPoolParallelism());
        Random random = new Random();
        int[] array = new int[2000000];
        ArrayList<Long> timeList = new ArrayList<>();
        for (int j = 50; j < 100; j++) {
            ParSort.cutoff = 40000 * (j + 1);
            // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
            long time;
            long startTime = System.currentTimeMillis();
            for (int t = 0; t < 10; t++) {
                for (int i = 0; i < array.length; i++) array[i] = random.nextInt( bound: 10000000);
                ParSort.sort(array,  from: 0, array.length);
            }
            long endTime = System.currentTimeMillis();
            time = (endTime - startTime);
            timeList.add(time);
```

```
Degree of parallelism: 7
cutoff：  2040000          10times Time:2261ms
cutoff：  2080000          10times Time:2083ms
cutoff：  2120000          10times Time:2079ms
cutoff：  2160000          10times Time:2066ms
cutoff：  2200000          10times Time:2065ms
cutoff：  2240000          10times Time:2062ms
cutoff：  2280000          10times Time:2057ms
cutoff：  2320000          10times Time:2061ms
cutoff：  2360000          10times Time:2069ms
cutoff：  2400000          10times Time:2074ms
cutoff：  2440000          10times Time:2066ms
cutoff：  2480000          10times Time:2052ms
cutoff：  2520000          10times Time:2057ms
cutoff：  2560000          10times Time:2060ms
cutoff：  2600000          10times Time:2062ms
cutoff：  2640000          10times Time:2044ms
cutoff：  2680000          10times Time:2057ms
cutoff：  2720000          10times Time:2077ms
cutoff：  2760000          10times Time:2079ms
cutoff：  2800000          10times Time:2079ms
cutoff：  2840000          10times Time:2067ms
cutoff：  2880000          10times Time:2062ms
cutoff：  2920000          10times Time:2074ms
cutoff：  2960000          10times Time:2068ms
cutoff：  3000000          10times Time:2057ms
cutoff：  3040000          10times Time:2066ms
```

```
cutoff：3080000              10times Time:2067ms
cutoff：3120000              10times Time:2061ms
cutoff：3160000              10times Time:2058ms
cutoff：3200000              10times Time:2062ms
cutoff：3240000              10times Time:2062ms
cutoff：3280000              10times Time:2051ms
cutoff：3320000              10times Time:2059ms
cutoff：3360000              10times Time:2066ms
cutoff：3400000              10times Time:2063ms
cutoff：3440000              10times Time:2067ms
cutoff：3480000              10times Time:2057ms
cutoff：3520000              10times Time:2069ms
cutoff：3560000              10times Time:2073ms
cutoff：3600000              10times Time:2064ms
cutoff：3640000              10times Time:2068ms
cutoff：3680000              10times Time:2067ms
cutoff：3720000              10times Time:2050ms
cutoff：3760000              10times Time:2073ms
cutoff：3800000              10times Time:2068ms
cutoff：3840000              10times Time:2081ms
cutoff：3880000              10times Time:2072ms
cutoff：3920000              10times Time:2064ms
cutoff：3960000              10times Time:2070ms
cutoff：4000000              10times Time:2073ms

Process finished with exit code 0
```

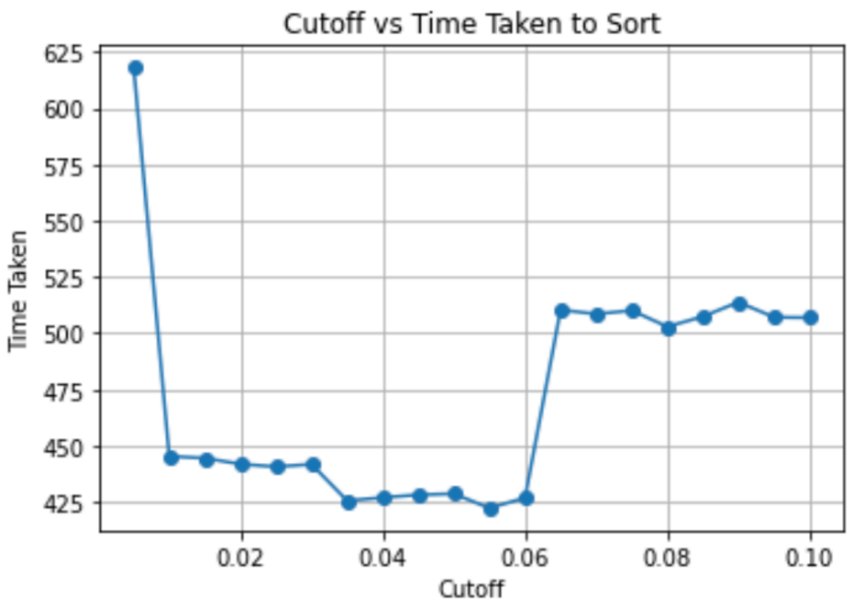
Degree of parallelism: 8
Array Size - 8M

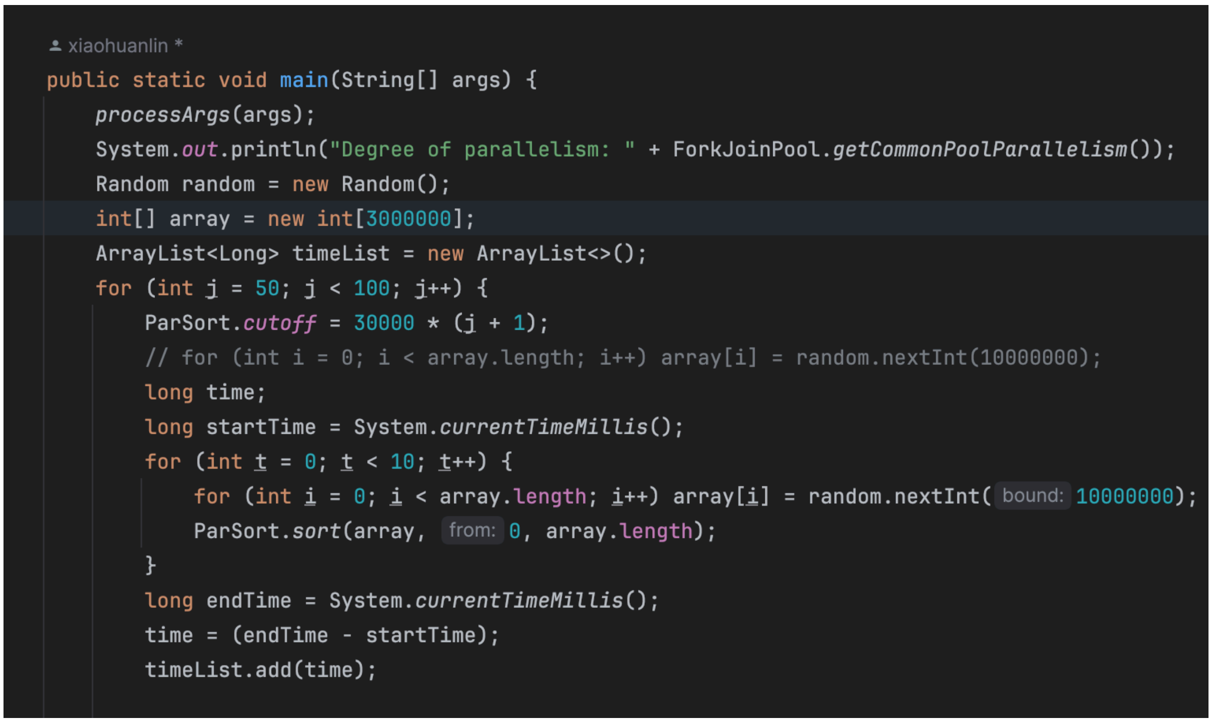


```
cutoff：200000               10times Time:5017ms
```

```
cutoff：400000                10times Time:3526ms
cutoff：600000                10times Time:3463ms
cutoff：800000                10times Time:3513ms
cutoff：1000000               10times Time:3489ms
cutoff：1200000               10times Time:3432ms
cutoff：1400000               10times Time:3387ms
cutoff：1600000               10times Time:3380ms
cutoff：1800000               10times Time:3395ms
cutoff：2000000               10times Time:3404ms
cutoff：2200000               10times Time:4013ms
cutoff：2400000               10times Time:4026ms
cutoff：2600000               10times Time:4049ms
cutoff：2800000               10times Time:4032ms
cutoff：3000000               10times Time:4040ms
cutoff：3200000               10times Time:4026ms
cutoff：3400000               10times Time:4045ms
cutoff：3600000               10times Time:4037ms
cutoff：3800000               10times Time:4025ms
cutoff：4000000               10times Time:4030ms

Process finished with exit code 0
```

```
Degree of parallelism: 8
Array Size 10M
cutoff：200000                10times Time:6182ms
cutoff：400000                10times Time:4453ms
cutoff：600000                10times Time:4444ms
cutoff：800000                10times Time:4418ms
cutoff：1000000               10times Time:4406ms
cutoff：1200000               10times Time:4417ms
cutoff：1400000               10times Time:4255ms
cutoff：1600000               10times Time:4269ms
cutoff：1800000               10times Time:4281ms
cutoff：2000000               10times Time:4286ms
cutoff：2200000               10times Time:4222ms
cutoff：2400000               10times Time:4268ms
cutoff：2600000               10times Time:5104ms
cutoff：2800000               10times Time:5085ms
cutoff：3000000               10times Time:5101ms
cutoff：3200000               10times Time:5029ms
cutoff：3400000               10times Time:5074ms
cutoff：3600000               10times Time:5138ms
cutoff：3800000               10times Time:5071ms
cutoff：4000000               10times Time:5069ms

Process finished with exit code 0
```



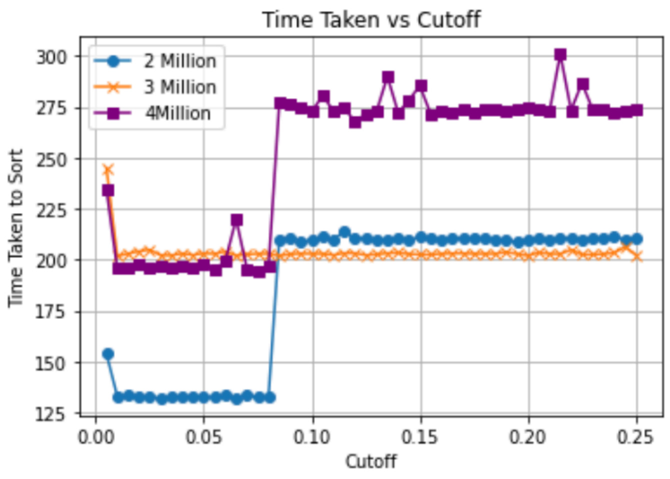


```
xiaohuanlin *
public static void main(String[] args) {
    processArgs(args);
    System.out.println("Degree of parallelism: " + ForkJoinPool.getCommonPoolParallelism());
    Random random = new Random();
    int[] array = new int[3000000];
    ArrayList<Long> timeList = new ArrayList<>();
    for (int j = 50; j < 100; j++) {
        ParSort.cutoff = 30000 * (j + 1);
        // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
        long time;
        long startTime = System.currentTimeMillis();
        for (int t = 0; t < 10; t++) {
            for (int i = 0; i < array.length; i++) array[i] = random.nextInt( bound: 10000000);
            ParSort.sort(array,  from: 0, array.length);
        }
        long endTime = System.currentTimeMillis();
        time = (endTime - startTime);
        timeList.add(time);
```

```
Degree of parallelism: 7
cutoff： 1530000            10times Time:2450ms
cutoff： 1560000            10times Time:2017ms
cutoff： 1590000            10times Time:2030ms
cutoff： 1620000            10times Time:2041ms
cutoff： 1650000            10times Time:2049ms
```

```
cutoff：1680000              10times Time:2022ms
cutoff：1710000              10times Time:2021ms
cutoff：1740000              10times Time:2030ms
cutoff：1770000              10times Time:2023ms
cutoff：1800000              10times Time:2031ms
cutoff：1830000              10times Time:2029ms
cutoff：1860000              10times Time:2040ms
cutoff：1890000              10times Time:2020ms
cutoff：1920000              10times Time:2026ms
cutoff：1950000              10times Time:2029ms
cutoff：1980000              10times Time:2029ms
cutoff：2010000              10times Time:2019ms
cutoff：2040000              10times Time:2027ms
cutoff：2070000              10times Time:2030ms
cutoff：2100000              10times Time:2029ms
cutoff：2130000              10times Time:2029ms
cutoff：2160000              10times Time:2023ms
cutoff：2190000              10times Time:2032ms
cutoff：2220000              10times Time:2032ms
cutoff：2250000              10times Time:2022ms
cutoff：2280000              10times Time:2026ms
cutoff：2310000              10times Time:2032ms
cutoff：2340000              10times Time:2034ms
cutoff：2370000              10times Time:2028ms
cutoff：2400000              10times Time:2025ms
cutoff：2430000              10times Time:2025ms
cutoff：2460000              10times Time:2029ms
cutoff：2490000              10times Time:2031ms
cutoff：2520000              10times Time:2031ms
cutoff：2550000              10times Time:2029ms
cutoff：2580000              10times Time:2029ms
cutoff：2610000              10times Time:2029ms
cutoff：2640000              10times Time:2038ms
cutoff：2670000              10times Time:2028ms
cutoff：2700000              10times Time:2020ms
cutoff：2730000              10times Time:2035ms
cutoff：2760000              10times Time:2029ms
cutoff：2790000              10times Time:2028ms
cutoff：2820000              10times Time:2049ms
cutoff：2850000              10times Time:2025ms
cutoff：2880000              10times Time:2025ms
cutoff：2910000              10times Time:2028ms
cutoff：2940000              10times Time:2034ms
cutoff：2970000              10times Time:2063ms
cutoff：3000000              10times Time:2024ms

Process finished with exit code 0
```

Graphical Representation



Trend in Time Taken: Generally, as the cutoff value increases, the time taken to sort tends to decrease initially and then fluctuates without a clear trend.

Variability in Time Taken: There is significant variability in the time taken to sort for different cutoff values. This variability suggests that other factors besides the cutoff value may also influence the sorting performance.

Optimal Cutoff Value: There does not appear to be a single optimal cutoff value that consistently minimizes the time taken to sort across all data points. The optimal cutoff value may vary depending on specific circumstances or characteristics of the dataset.