

Nishay Madhani (00212195)

# Program Structures & Algorithms

Fall 2021

## Assignment No. 5

### Tasks Performed

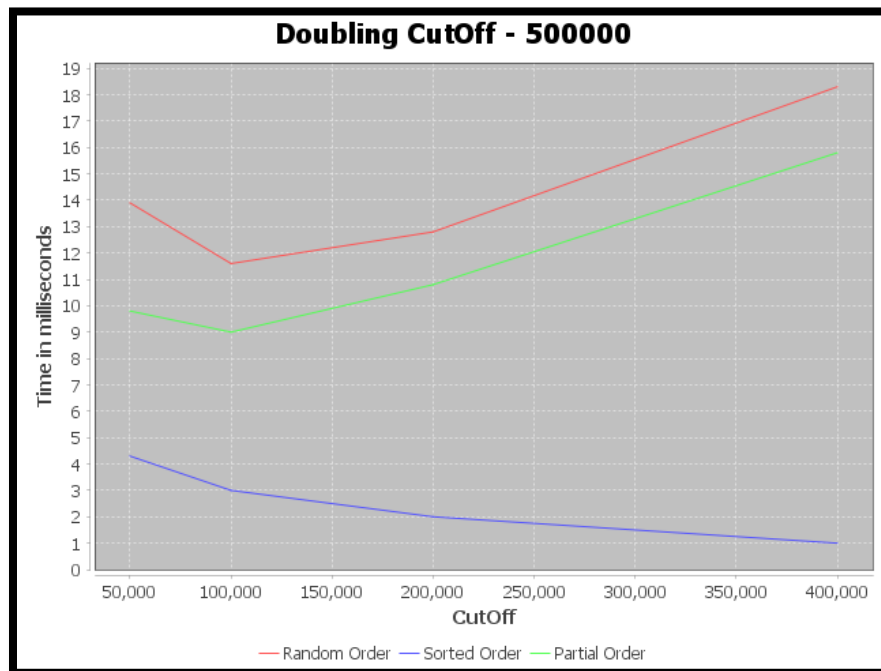
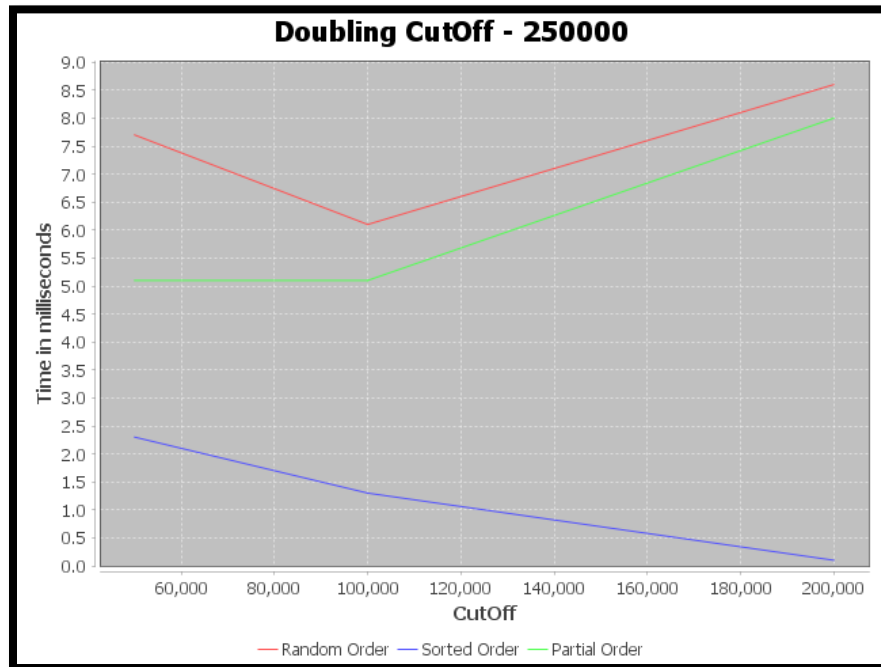
1. Created a CustomOrdering utility class as helper to give arrays in different orderings such as:
  - a. Partial
  - b. Sorted
  - c. Random
2. Modified the PartSort class with the ability to modify the cutOff value and the number of threads.
  - a. Created an ExecutorService with a built in ThreadFactory to be able to name each thread for debugging purposes.
  - b. Added an additional else clause to the sort() function to use the system sort when no thread is available in the threadpool and we are only collecting data for the number of threads and not cutoff.
  - c. Removed dependency of all methods being static into member methods to be able to sort algos and change cutoff values and threads frequently.
3. Added a few test cases to test the correctness of the algorithm
4. Added a scheme interface called Scheme with one next() function to use different kinds of schemes to update values for threads and cutoff
  - a. Incremental Scheme - incremented by 10% of current value
  - b. Doubling Scheme - multiplied by two in each iteration
5. Case 1: Cutoff Schemes
  - a. case1ForCutOffSchemes()
    - i. Sorts a particular array and updates the cutoff with a given scheme
    - ii. Uses the Benchmark class used in an earlier assignment to time the algorithm.
    - iii. Returns the time taken to sort for each sort
  - b. runCutOffCase()

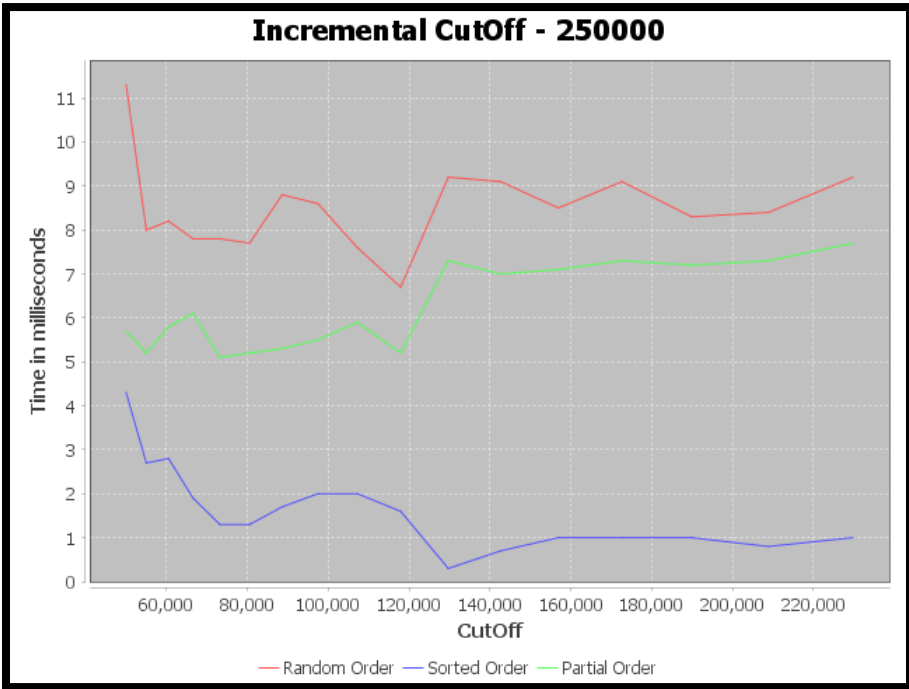
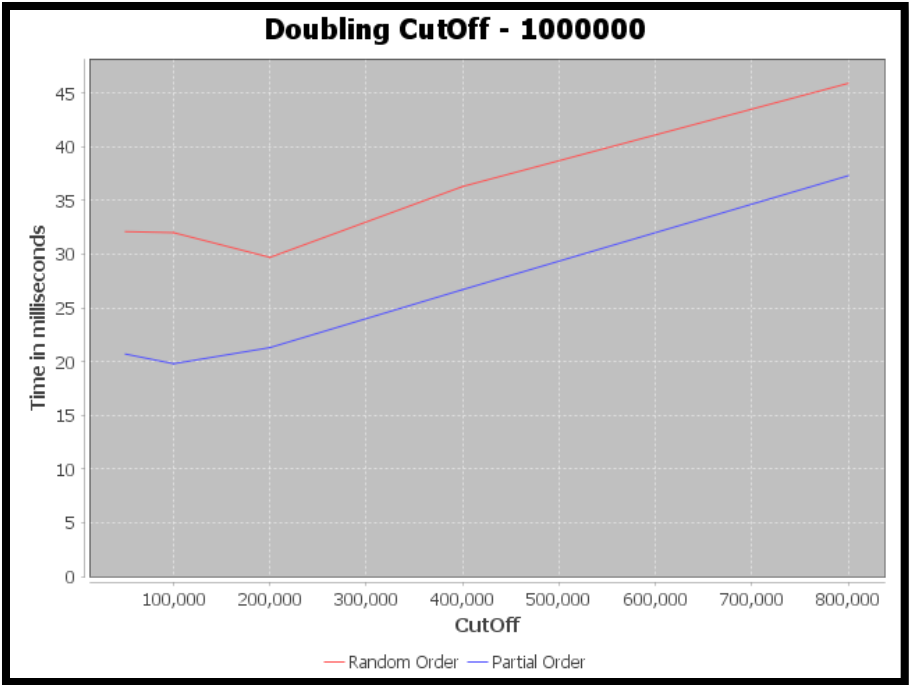
- i. Uses the previous function to sort different array orderings: sorted, partial, random.
    - ii. Collects all the data and creates a chart using it.
  - c. runCutOffCases()
    - i. Uses the previous function
      - 1. Multiple arrays sizes
      - 2. Multiple cutoff updation schemes
6. Case 2: Thread Schemes
- a. case1ForThreadSchemes()
    - i. Sorts a particular array and updates the number of threads with a given scheme
    - ii. Uses the Benchmark class used in an earlier assignment to time the algorithm.
    - iii. Returns the time taken to sort for each sort
  - b. runThreadCase()
    - i. Uses the previous function to sort different array orderings: sorted, partial, random.
    - ii. Collects all the data and creates a chart using it.
  - c. runThreadCases()
    - i. Uses the previous function
      - 1. Multiple arrays sizes
      - 2. Uses Doubling method to update the number of threads
7. Case 3: Combination of Case 1 and Case 2
- a. Used a combination of values obtained from the above cases to find the best combination

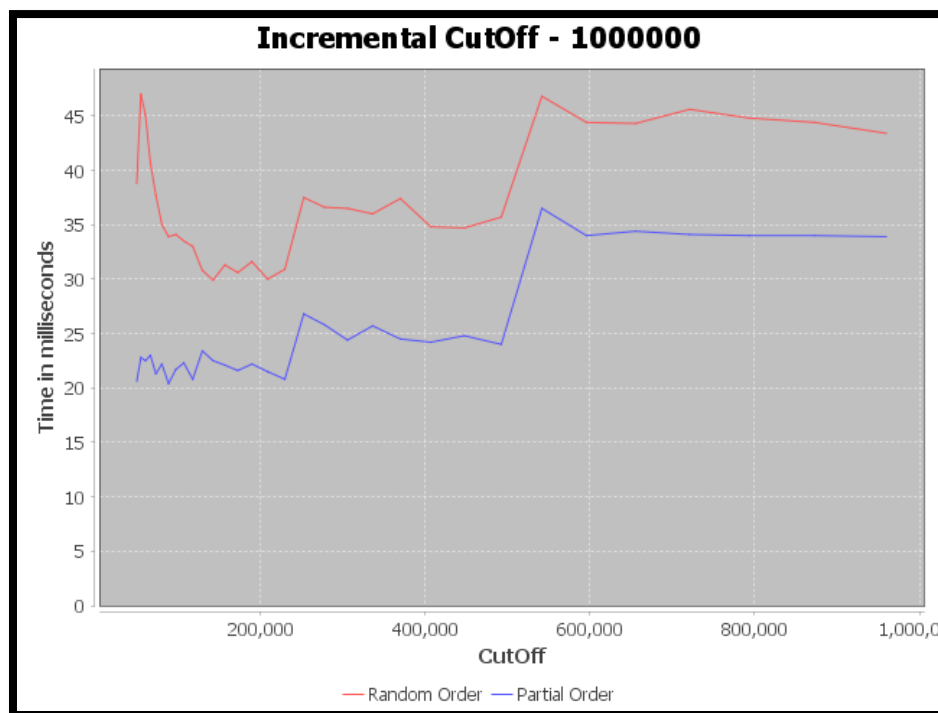
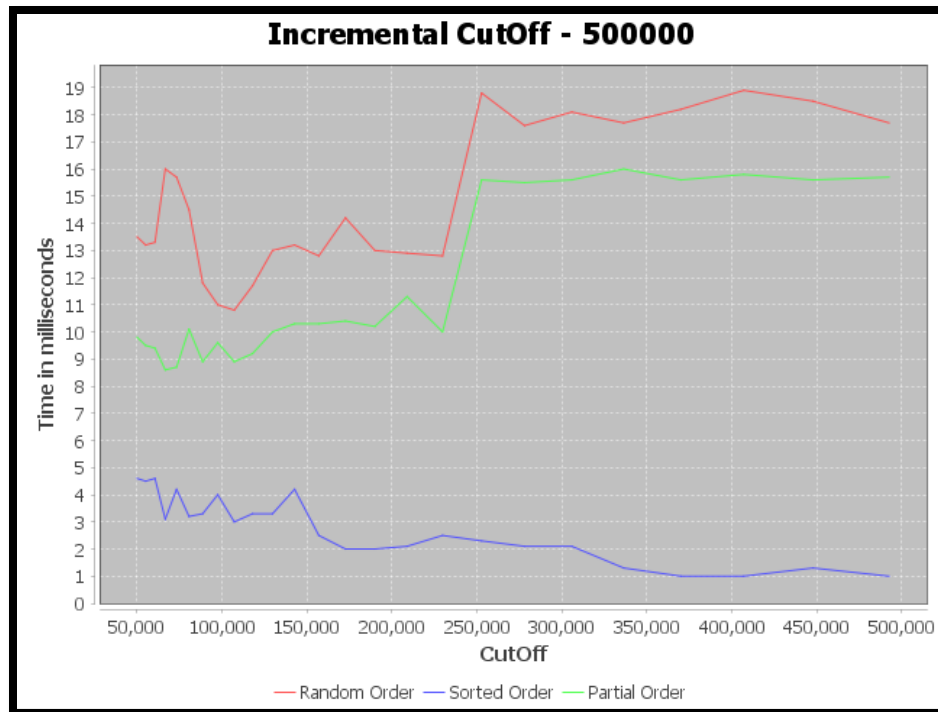
## Conclusion

<b>Best Cutoff Value = 10-15% of N</b>
<b>Best Number of Threads = 500 to 1000 for N &gt; 1,000,00</b>
<b>20 to 100 for N &lt;= 100,00</b>

## Observation: Cutoff Schemes





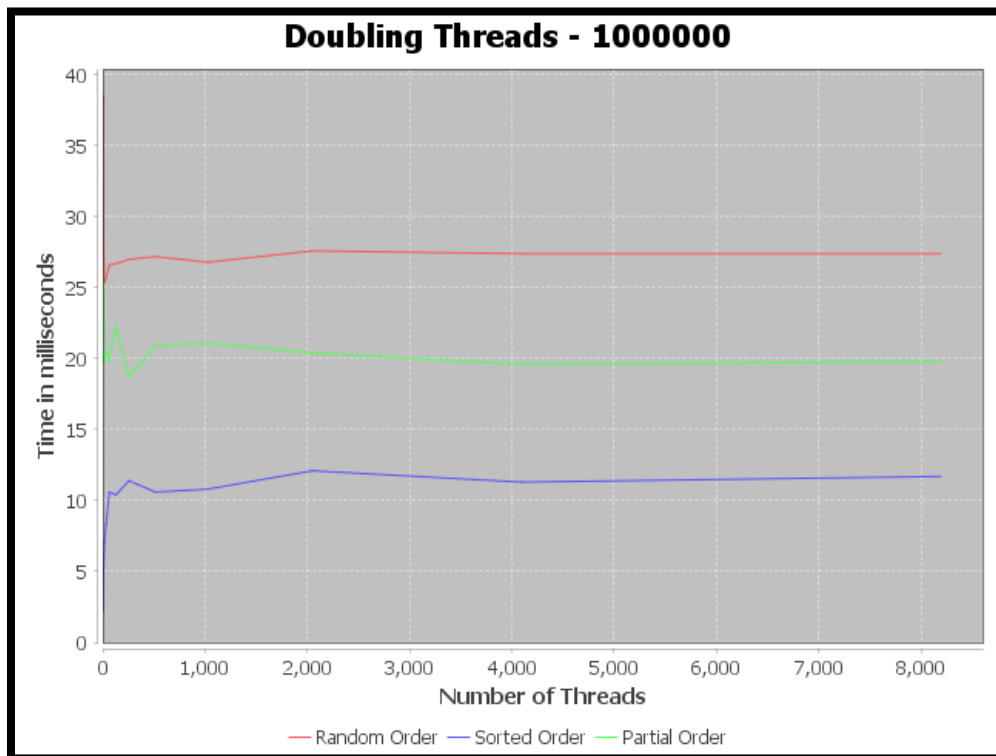
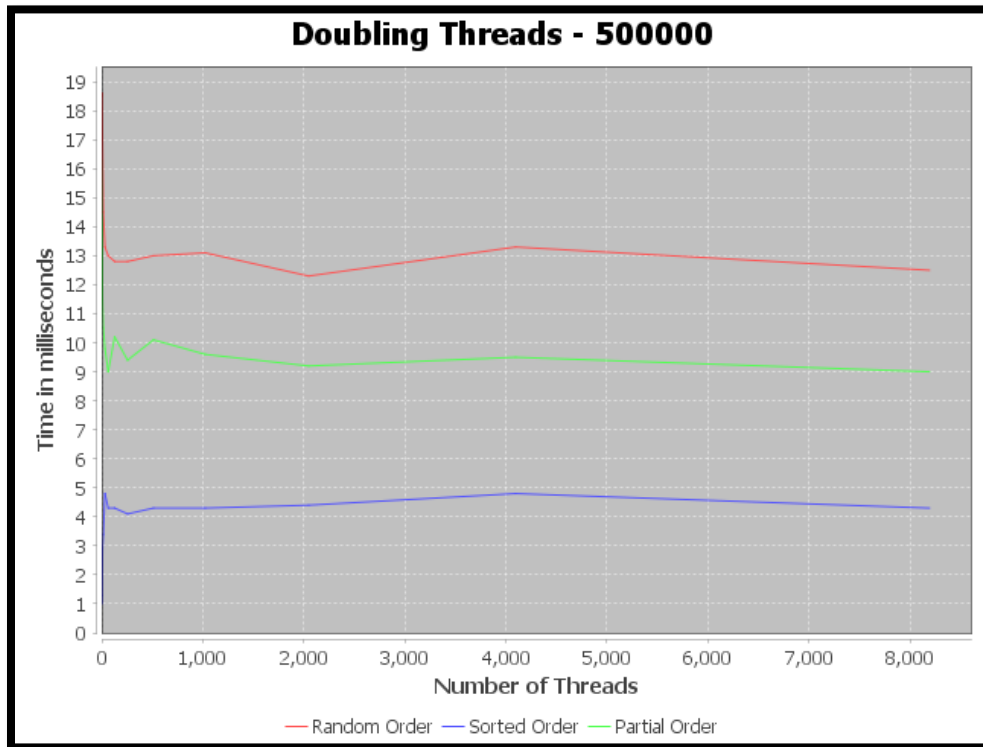


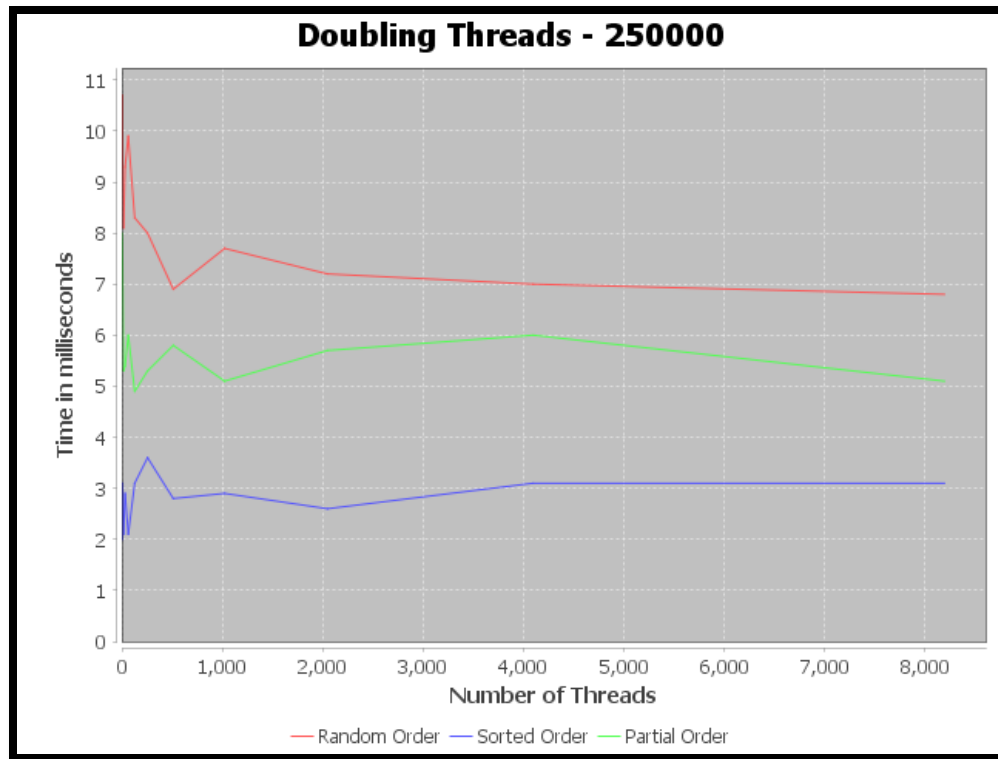
- **Note:**
  - For over a million elements Partial ordering was not used as the system would throw an Exception regarding the lack of resources

- Results:
  - For each of the different types of orderings we can infer that as the cutoff increases over time, the total time taken to run the algorithm also increases.
  - This phenomenon can be attributed to using system sort when the length of array is less than or equal to the cutoff. Such a behavior for very large values of N will make the system increasingly “non-parallel” or “sequential” as the threads used will reduce so will the concurrency of the system.
- As denoted by the graphs for each of the different orderings, there is a significant dip in running time as the cutoff increase but then such a progression turns out to be detrimental to the algorithm itself.

<b>Best Cutoff Value = 100,000</b>
<b>Best Cutoff Percentage Range = 15-20% of total elements</b>

## Observation: Number of Threads



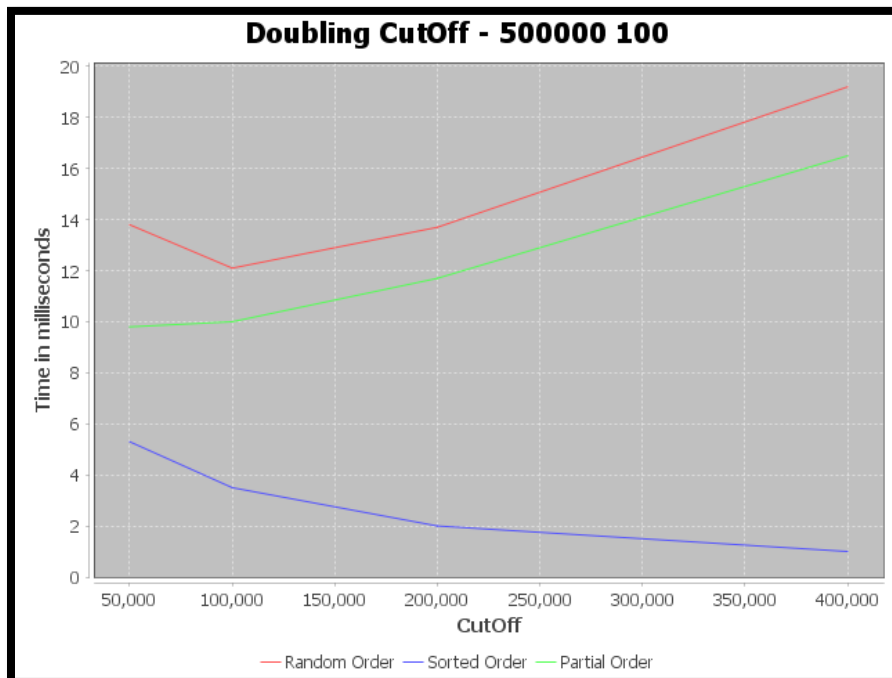
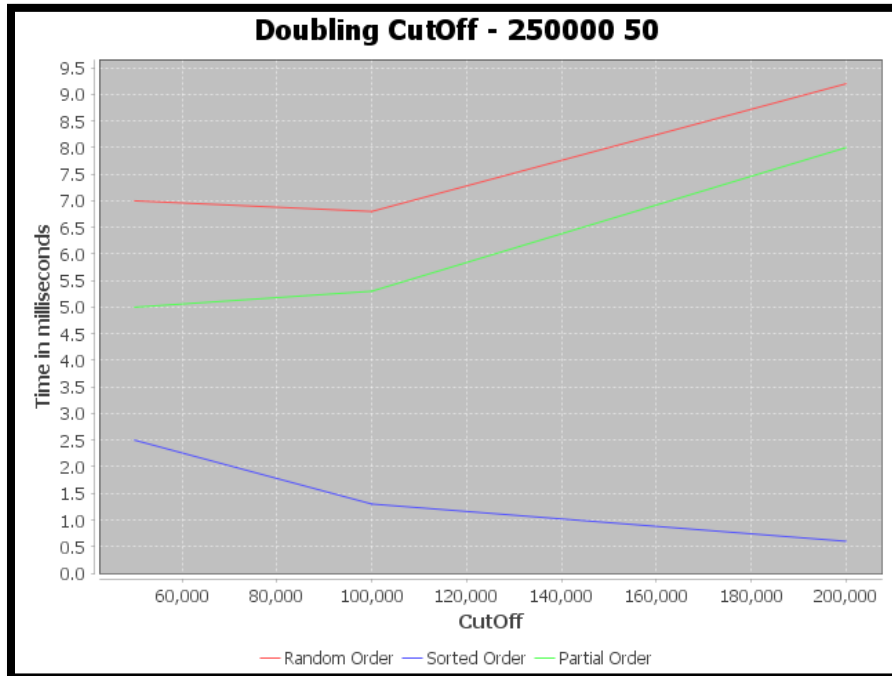


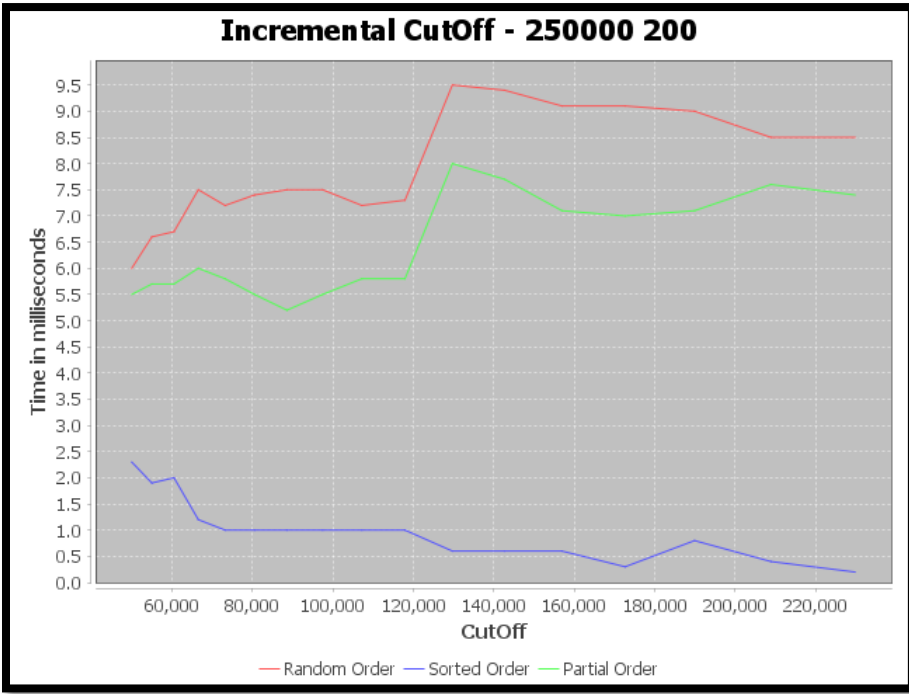
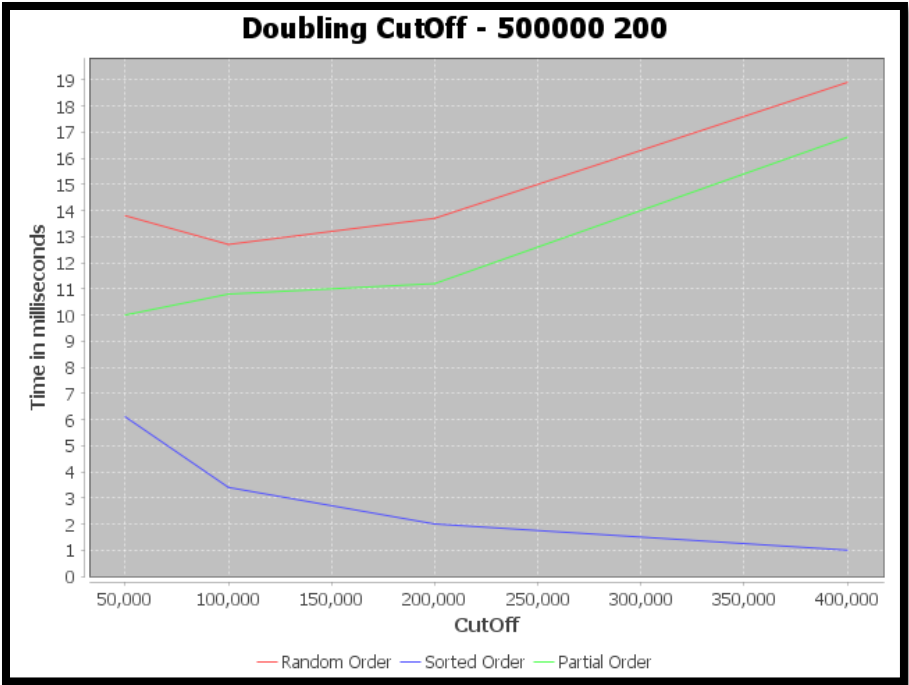
- **Notes:** To carry out this experiment I created a [ExecutorService](#) and changed the number of threads used by every iteration. Number of threads used is therefore limited by the number of physical cores and threads available.
- The graph shows a straight line as the number of threads available to the algorithm increases.
  - This behavior is to be expected from the algorithm as the number of threads actually required by the algorithm will be bounded by the size of its divide and conquer tree, which is  $2^{(\log N)}$ .
  - However, the algorithm will require even less threads than the upper bound because as each thread finishes its work and returns to the pool, it can be easily reused by another *parsort* process.
- Apart from the sudden decline as the number of threads starts doubling from 2, we can see a steady line as the algorithm does not require any more threads.

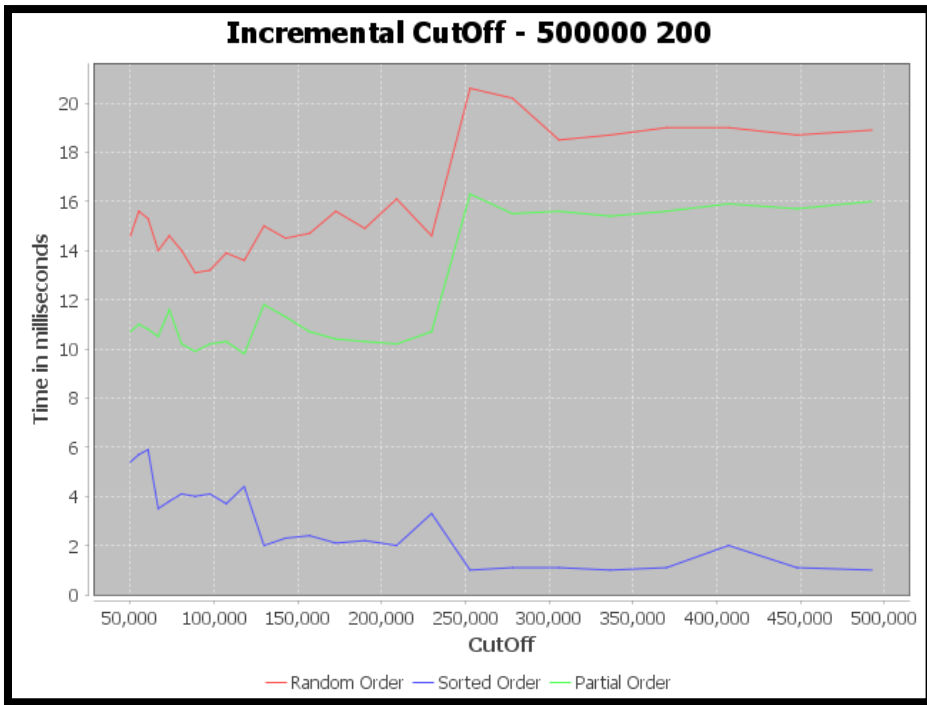
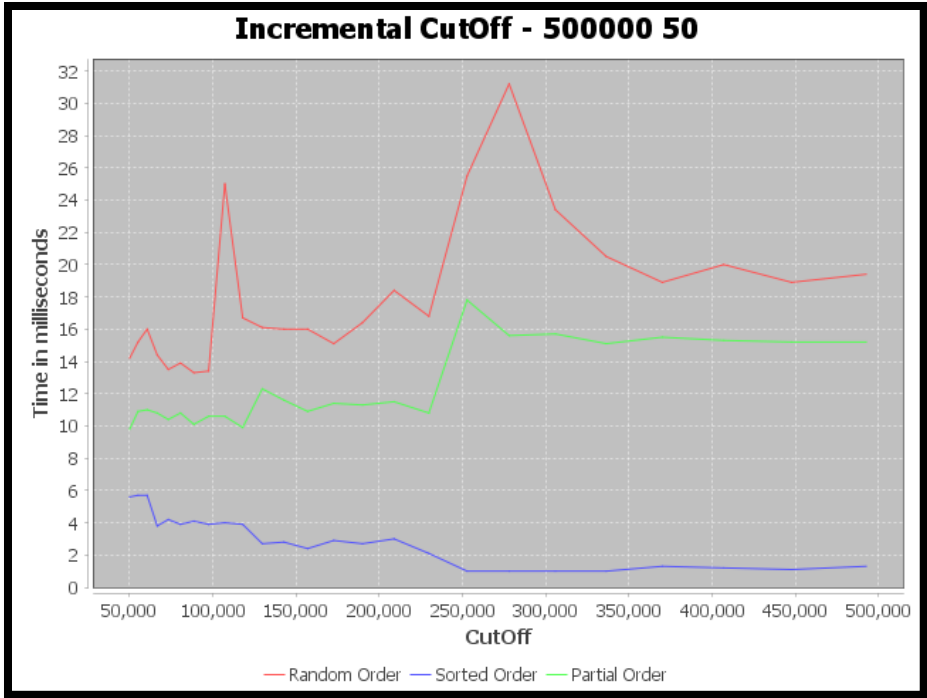
**Optimum Number of threads = 500-1000**

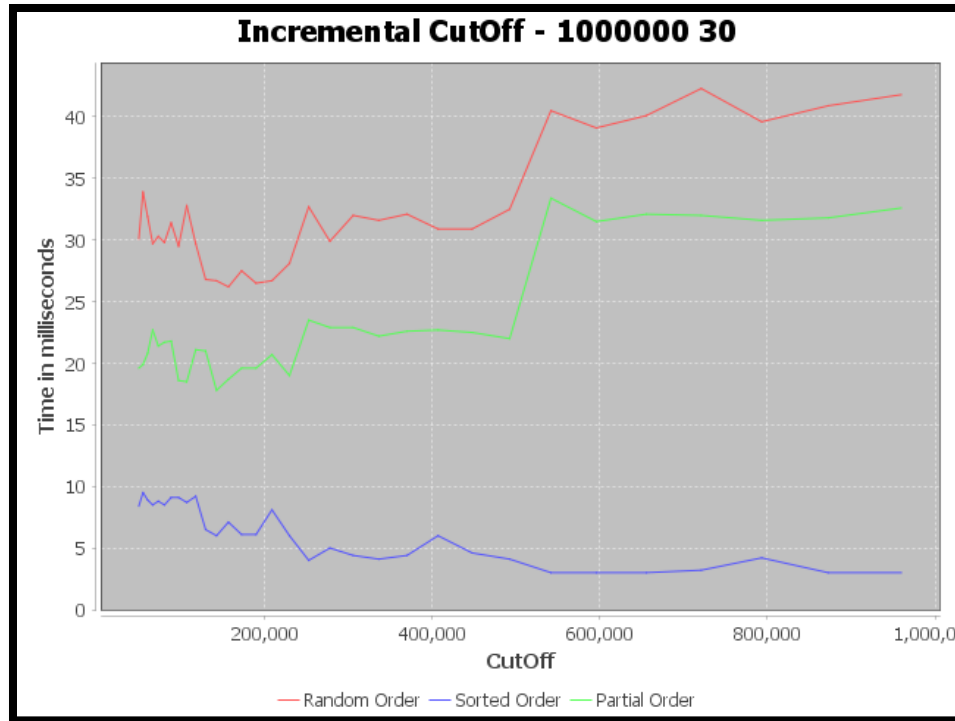


## Observation: Combination







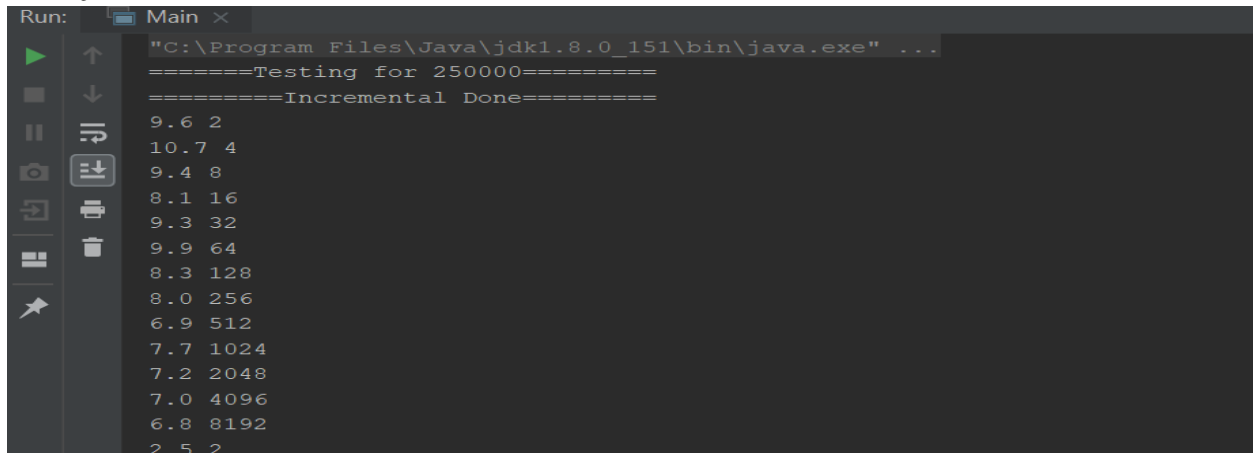


- **Notes:** To test combination of the two schemes a selection of the best number of threads was taken along with the varying values of cutoff based on different schemes
  - To find the best number of threads, a total of 20 experiments were done but only some of the results have been published above
- As expected, the graph follows the same kind of progression seen before in both experiments, the value of the cutoff increases too much it makes the algorithm “serial”. If the number of threads is too large It ill have no effect on the running time as the number of threads wont be required.

<b>Best Cutoff Value: 200,00</b>
<b>Best Number of Threads: 200</b>

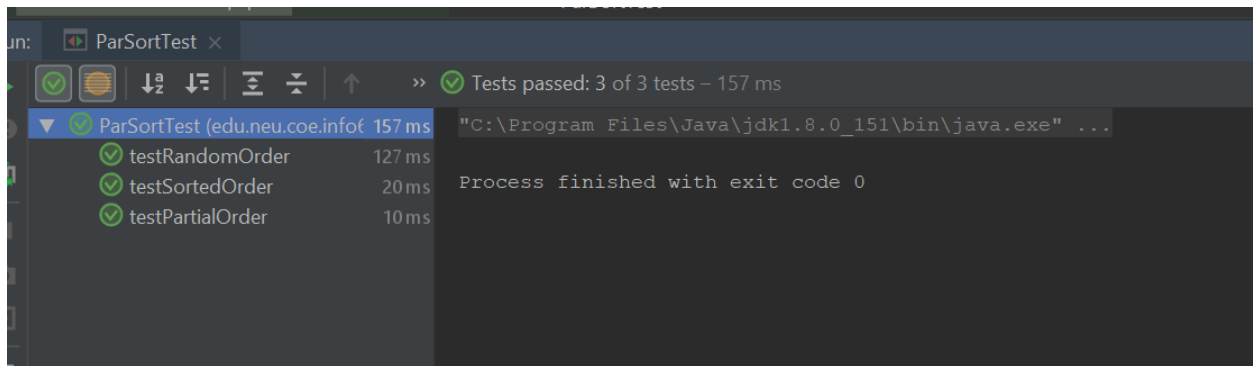
# Unit Tests and Output

## *Main.java*



```
Run: Main x
"C:\Program Files\Java\jdk1.8.0_151\bin\java.exe" ...
=====Testing for 250000=====
=====Incremental Done=====
9.6 2
10.7 4
9.4 8
8.1 16
9.3 32
9.9 64
8.3 128
8.0 256
6.9 512
7.7 1024
7.2 2048
7.0 4096
6.8 8192
2.5 2
```

## *ParSortTest.java*



```
Run: ParSortTest x
Tests passed: 3 of 3 tests - 157 ms
ParSortTest (edu.neu.coe.infof 157 ms)
  testRandomOrder 127 ms
  testSortedOrder 20 ms
  testPartialOrder 10 ms
"C:\Program Files\Java\jdk1.8.0_151\bin\java.exe" ...
Process finished with exit code 0
```