# *Interval Partitioning Report*

*Mark Dear, Stelios Papaoikonomou, Radek Niemczyk, Rasmus Løbner Christensen*

*September 9, 2015*

## *Results*

Our implementation produces the expected results on all input–output file pairs.

Our own JUnit test, which we are not including in the submission, showed that we have a total of N loops in the main for loop (in the Algorithm implementation). This ensures us that the upper bound on the runtime, is bound by the Java Array sort. Looking at the documentation for Comparable.compareTo() complexity, tells us that this runs in O(n log n), which means that our implementation is running at a complexity of O(n log n).

## *Implementation details*

Our implementation takes the input file and saves this in a String[] (fileParser.java). This String[] is then parsed through our parser class (parser.java), which delegates the data into our dataobjects. Our actual Algorithm implementation starts with sorting the list of Jobs[] (the intervals). Then it loops through each of these, and checks for available resources (partitions). The partitions are stored in a Java priorityQueue, which always have the partition with the earliest available endtime at the first element. This makes it possible for us to not check through the whole list of partitions, for each n (input/interval). If the partition is occupied, it creates a new partition, and put this correctly into our Queue. Lastly we print the output to the console.