Group 4: Isaac Supeene and Braeden Soetaert

No design changes to mention but there are some important design choices that are important to note.

Our sort does the sort in place so if our sort times out or experiences an error, the array of objects that we are passed may be rearranged, but not completely sorted. We do the sorting in place because the array can be very large, it's probably not rational to allocate enough space for a copy of the array.

If our program experiences an error in one of its threads, that thread will be retried once. If the error is still present, then the program cleans up by cancelling its other threads and then raises the relevant error so the caller will receive this error. In this way, the caller of our sort will be able to see the relevant error rather than needing to modify our code if we had instead just returned false.

To do the sort in place, we use an in-place parallel merge algorithm described here. This algorithm in turn uses a parallel algorithm for block exchange, using the array reversal method. This algorithm was chosen for block exchange because of its high parallelism.

With respect to our contract and the provided timeout, we provide ourselves with a buffer of 1 second to cleanup after the timeout has expired.

We are using Ruby version 1.9.3 for our coding. It may work on other versions of Ruby but there are no guarantees.