Group Project 2

With each of these questions you should:

- **Do examples by hand** – make up some example input and run the algorithms by hand first to make sure you understand how they work.
- Get simple things working before attempting to optimize or make cleverer programs!
- Sketch your programs in **pseudo-code** first, before coding them.
- **Document your message protocol** – what messages are sent, their format, and the expected reply if any.

You should submit your examples, your pseudo-code, your message protocol, and your Java code for each question.

Extra Requirements:

- Your code must use "UDP-style" messages over TCP. By this I mean use only one half of a socket. So, given a socket **S**, either use **S's** output stream to send a message or **S's** input stream to read a message, but **NEVER** read and write from the same socket. So you are still using TCP, but sending and receiving messages as if they are UDP messages, which is more realistic.
- You can write to the output stream of a socket using **PrintWriter**, and read from it using a **BufferedReader**. These do work but are **NOT** the most efficient way of transferring data over a socket. You should use an efficient way of transferring data.

Question 1

Write a **distributed** Java program to find the **k-th** smallest number in a non-ordered **array** of numbers.

The program will start with the array of numbers present at one node, but not at the others.

Each node should run a **sequential** program – so you have a network of sequential programs. The simplest organization is to have a co-ordinator node and **N** partitioning nodes.

You should use TCP connections between nodes to simplify things, and have at least 3 nodes in your network (so three separate computers).

Note: If you don't have access to 3 separate computers you can run more than one node on one machine by having each “node” listen to different TCP ports. This entails having one main method for each node. To avoid code duplication your main method should simply create a server socket on the appropriate port (which varies from node to node), and then call a **shared** static method (which does not vary from port to port) in some other class, passing the server socket previously created. This will minimize the amount of copy pasting you have to do to make multiple main methods.

Question 2

Modify your program, and potentially your message protocol, from question 2, so that each node runs a parallel program – so you have a network of parallel programs. Again the simplest organization is to have a co-ordinator node and **N** partitioning nodes, but now the partitioning nodes must be parallel programs.