Results hand-in.

The following table summarises our results. It shows the average number of random connections needed before the emergence of the giant component (“giant”), the disappearance of the last isolated individual (“no isolated”), and when the network becomes connected (“connect”).

N	T	Giant	(stddev)	no isolated	(stddev)	connected	(stddev)
100	1000000	73.1	6.8	259.4	61.8	259.4	61.8
1000	10000	697.0	17.7	3752.6	647.0	3752.6	647.0
10^4	10000	6935.7	55.0	48941.8	6547.8	48941.8	6547.8
10^5	10000	69319.2	174.2	604483.4	64362.0	604483.4	64362.0
10^6	1000	693162.3	557.3	7190827.9	643538.4	7190827.9	643538.4
10^7	10	6931324.1	1890.1	8.2×10^7	5098419.9	8.2×10^7	5098419.9

Our main findings are the following: The first thing that happens is that the giant component emerges, which almost happens at a time linear in N (which we found when plotting the data into a graph and also we see that the slope is approximately the same for all values of Giant, namely 10). Perhaps surprisingly, two of the events seem to happen simultaneously, namely the non-isolated state and the connected state, which also happen at a time linear in N.

Implementation details

We have based our union-find data type on WeightedQuickUnionFind.java from Sedgwick and Wayne: Algorihthms, 4th ed., Addison–Wesley (2011). We added the methods setNonIsolated(), setGiant() and setConnected() plus getter methods for boolean fields isNotIsolated, isGiant and isConnected.
Assuming we never run out of memory or heap space, if we would let our algorithm for detecting the emergence of a giant component run for 24 hours, it could compute the answer for around
$N = 5{,}27 * 10^{11}$. (estimated by using “Stopwatch” to find average time for when the giant component emerges and use this time to estimate N for 24 hours).