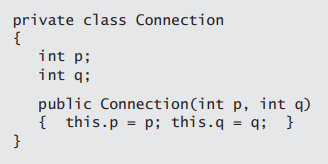
1.5.20 Dynamic growth. Using linked lists or a resizing array, develop a weighted quick-union implementation that removes the restriction on needing the number of objects ahead of time. Add a method newSite() to the API, which returns an int identifier.



Solution: Attached file (DynamicGrowth.java)

1.5.21 Erdös-Renyi model. Use your client from Exercise 1.5.17 to test the hypothesis that the number of pairs generated to get one component is ~ ½N ln N.

Solution: Attached file (ErdosRenyi.java)

1.5.22 Doubling test for Erdös-Renyi model. Develop a performance-testing client that takes an int value T from the command line and performs T trials of the following experiment: Use your client from Exercise 1.5.17 to generate random connections, using UnionFind to determine connectivity as in our development client, looping until all sites are connected. For each N, print the value of N, the average number of connections processed, and the ratio of the running time to the previous. Use your program to validate the hypotheses in the text that the running times for quick-find and quick-union are quadratic and weighted quick-union is near-linear.

Solution: Attached file (ErdosRenyiDoublingTest.java)

1.5.23 Compare quick-find with quick-union for Erdös-Renyi model. Develop a performance-testing client that takes an int value T from the command line and performs T trials of the following experiment: Use your client from Exercise 1.5.17 to generate random connections. Save the connections, so that you can use both quick-union and quick-find to determine connectivity as in our development client, looping until all sites are connected. For each N, print the value of N and the ratio of the two running times.

Solution: Attached file (UFPerformanceTest.java)

1.5.24 Fast algorithms for Erdös-Renyi model. Add weighted quick-union and weighted quick-union with path compression to your tests from Exercise 1.5.23 . Can you discern a difference between these two algorithms?

Solution: Attached file (UFPerformanceTest.java)

1.5.25 Doubling test for random grids. Develop a performance-testing client that takes an int value T from the command line and performs T trials of the following experiement: Use your client from Exercise 1.5.18 to generate the connections in an N-by-N square grid, randomly oriented and in random order, then use UnionFind to determine connectivity as in our development client, looping until all sites are connected. For each N, print the value of N, the average number of connections processed, and the ratio of the running time to the previous. Use your program to validate the hypotheses in the text that the running times for quick-find and quick-union are quadratic and weighted quick-union is near-linear. Note : As N doubles, the number of sites in the grid increases by a factor of 4, so expect a doubling factor of 16 for quadratic and 4 for linear.

Solution: Attached file (GridUnionFindPerformanceTest.java)