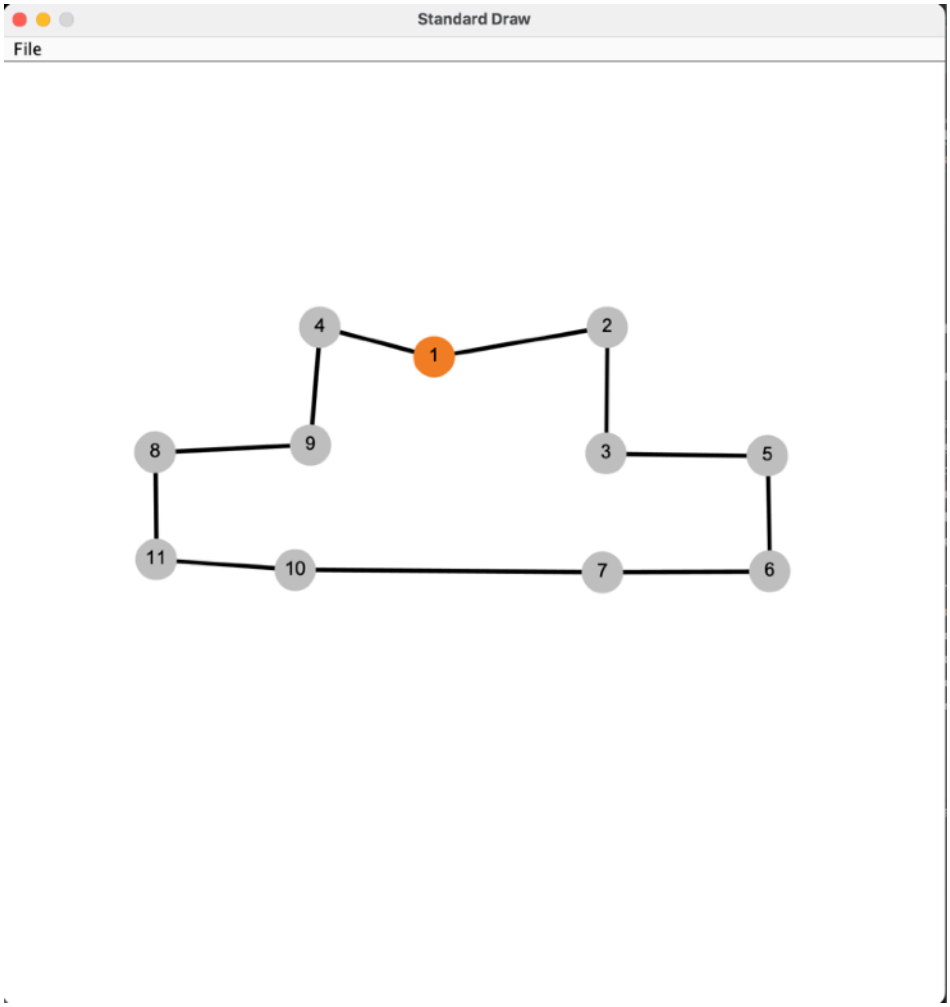
Section 1

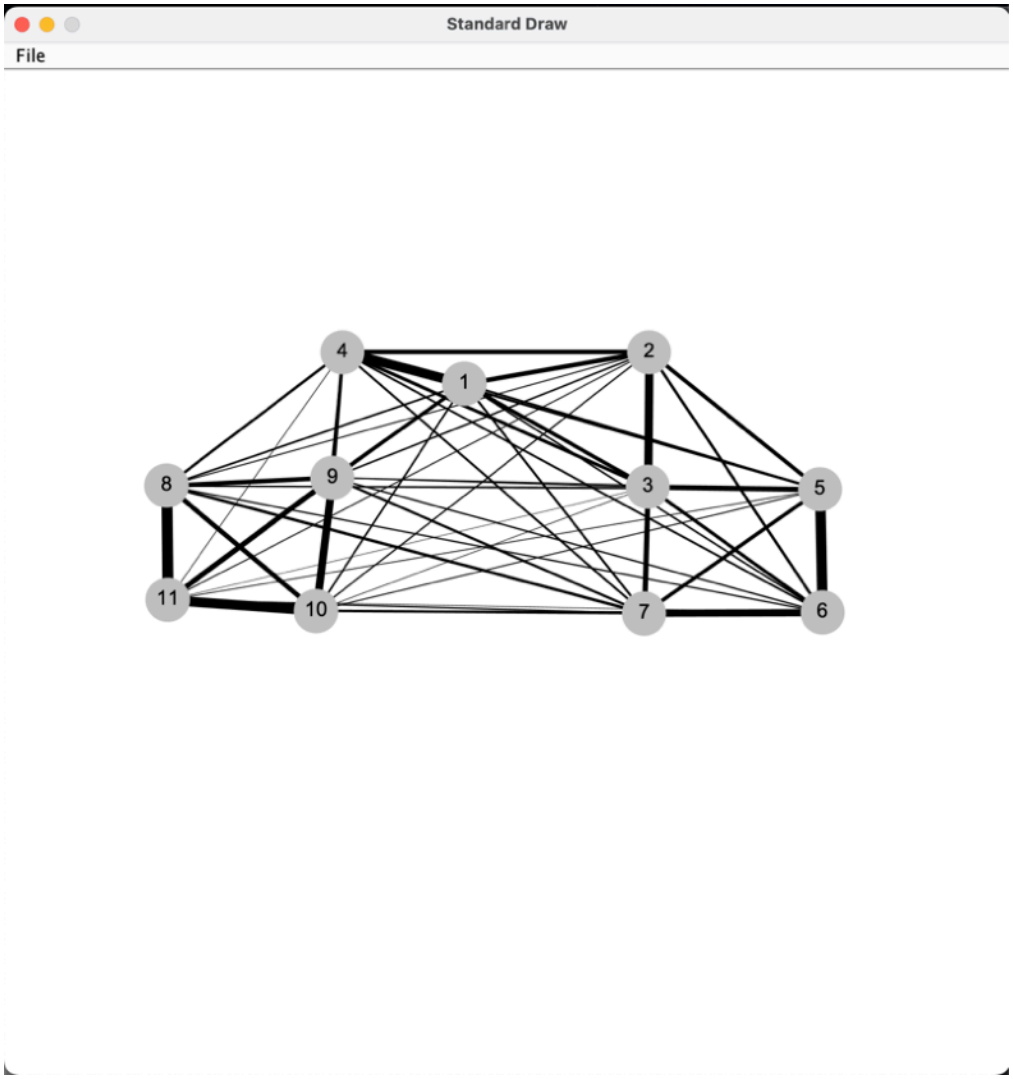
Outputs of different approaches

Input1

Shortest route found by both the bruteForce and antColony methods are the same for this approach.



The pheromone density map for antColony:



The outputs of said approaches are below. As seen, antColony performs much faster than bruteForce even for this smaller output.

```
Method: Brute-Force Method
Shortest Distance: 1.79529
Shortest Path: [1, 4, 9, 8, 11, 10, 7, 6, 5, 3, 2, 1]
Time it takes to find the shortest path: 2.91 seconds.
```

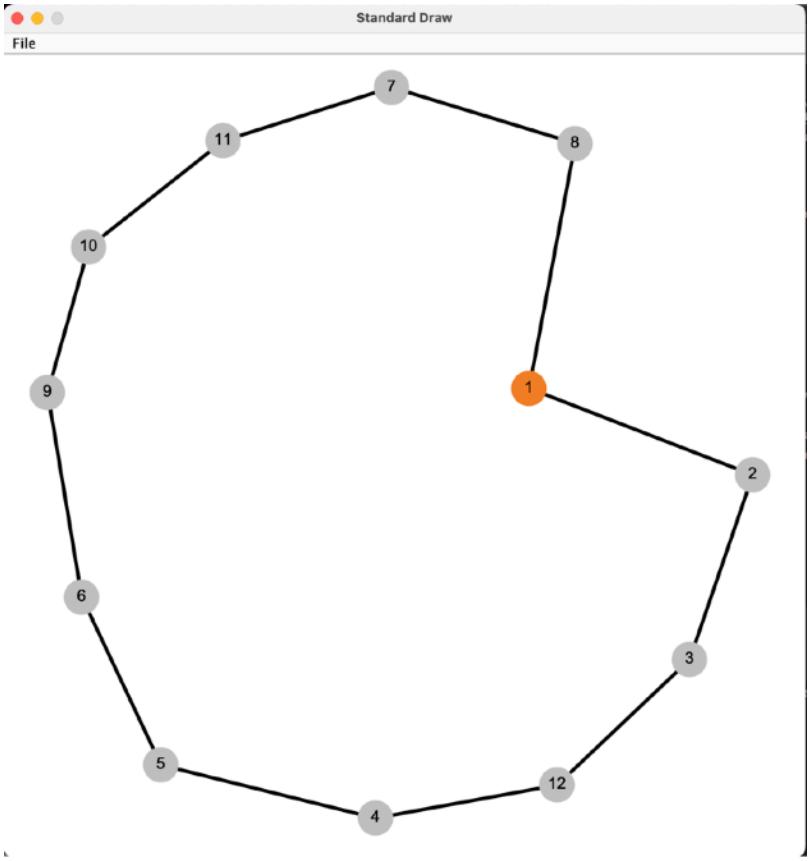
```
Method: Ant-Colony Method
Shortest Distance: 1.79529
Shortest Path: [1, 4, 9, 8, 11, 10, 7, 6, 5, 3, 2, 1]
Time it takes to find the shortest path: 0.13 seconds.
```

The ‘best’ parameters used for ACO in input1 are the same as suggested in the project outline (except for initial pheromone levels, with slightly decremented BETA), being:

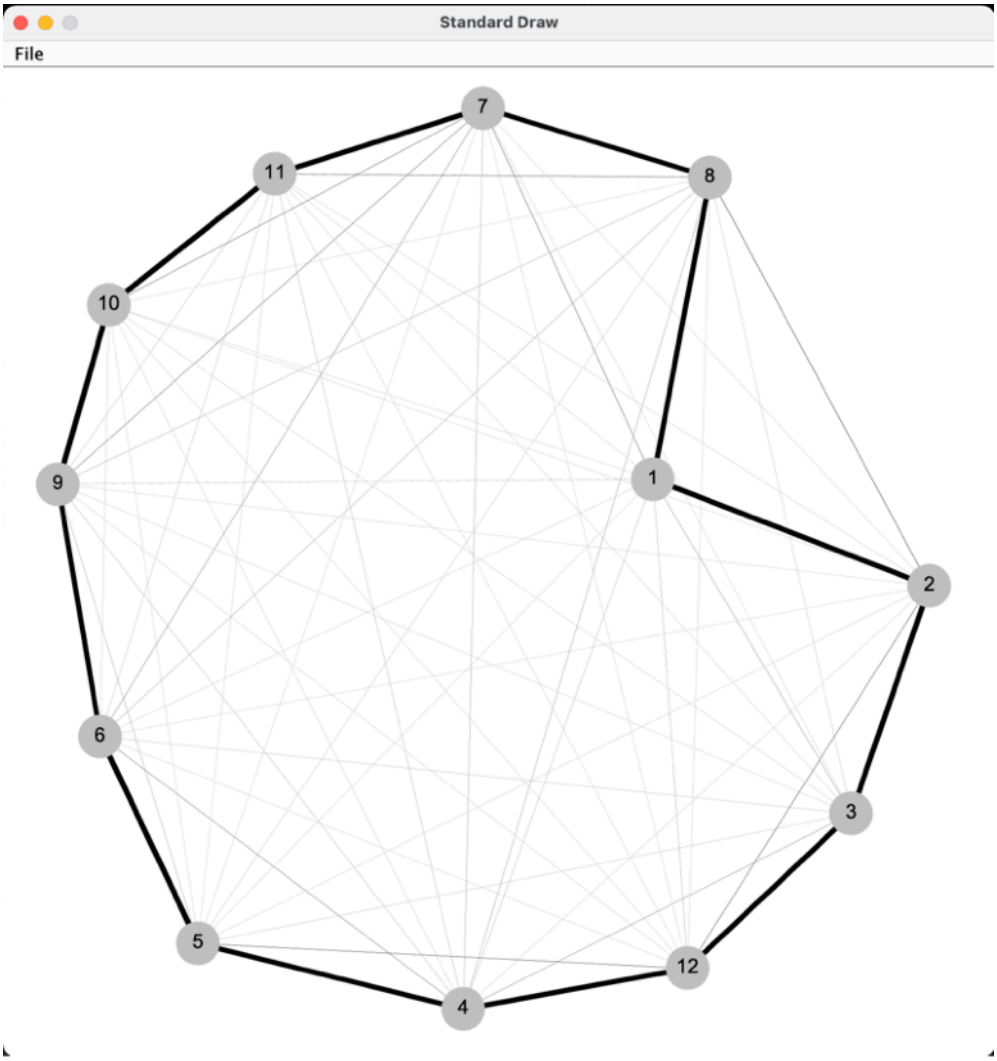
```
// Constants to be used in ACO algorithm
final int iterationCount = 100; // Count of iterations
final int antCount = 50; // Size of colonies to be iterated over
final double degradationFactor = 0.9; // Evaporation rate
final double ALPHA = 0.8; // Alpha constant
final double BETA = 1.2; // Beta constant
final double initialPheromone = 0.001; // Initial pheromone on roads
final double Q = 0.0001; // Q constant
```

Input2

Shortest route found by both the bruteForce and the antColony methods are the same for this approach.



The pheromone density map for antColony:



The outputs of said approaches are below. As seen, antColony performs even faster compared to bruteForce this time.

```
Method: Brute-Force Method
Shortest Distance: 2.93588
Shortest Path: [1, 8, 7, 11, 10, 9, 6, 5, 4, 12, 3, 2, 1]
Time it takes to find the shortest path: 41.95 seconds.
```

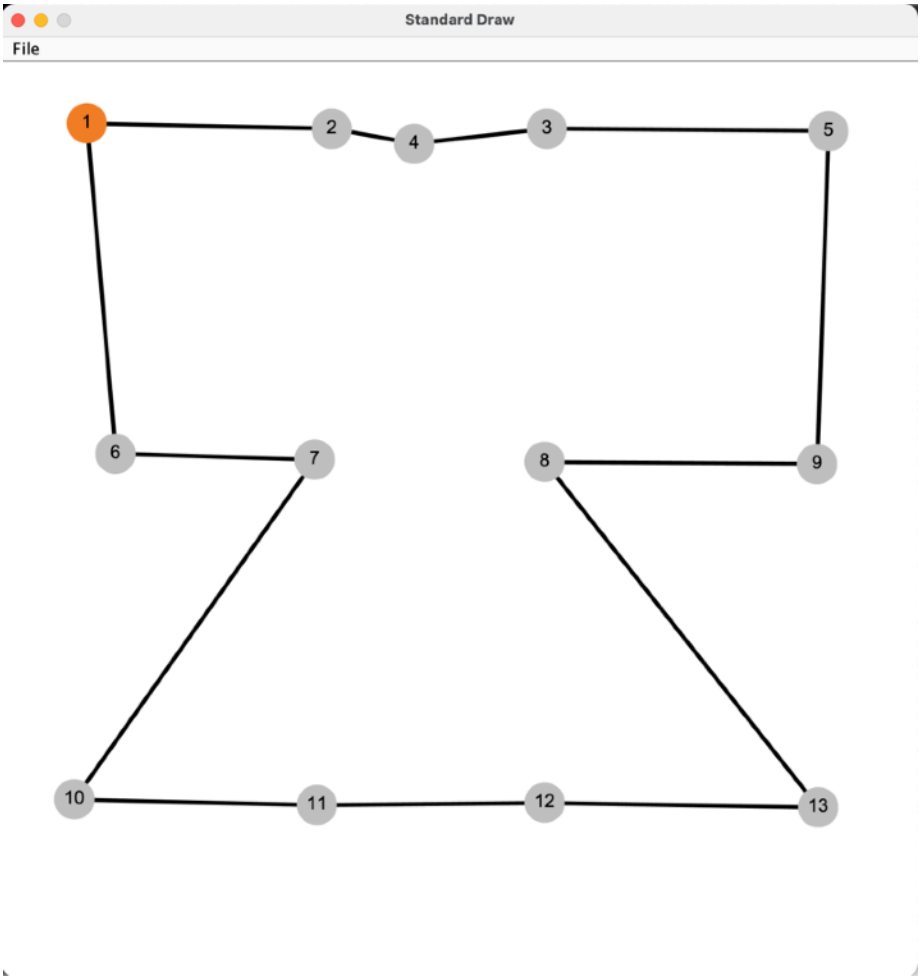
```
Method: Ant-Colony Method
Shortest Distance: 2.93588
Shortest Path: [1, 2, 3, 12, 4, 5, 6, 9, 10, 11, 7, 8, 1]
Time it takes to find the shortest path: 0.17 seconds.
```

However, the 'best' parameters for antColony differ from the suggested parameters here, as the suggested parameters can give a worse result seldom.

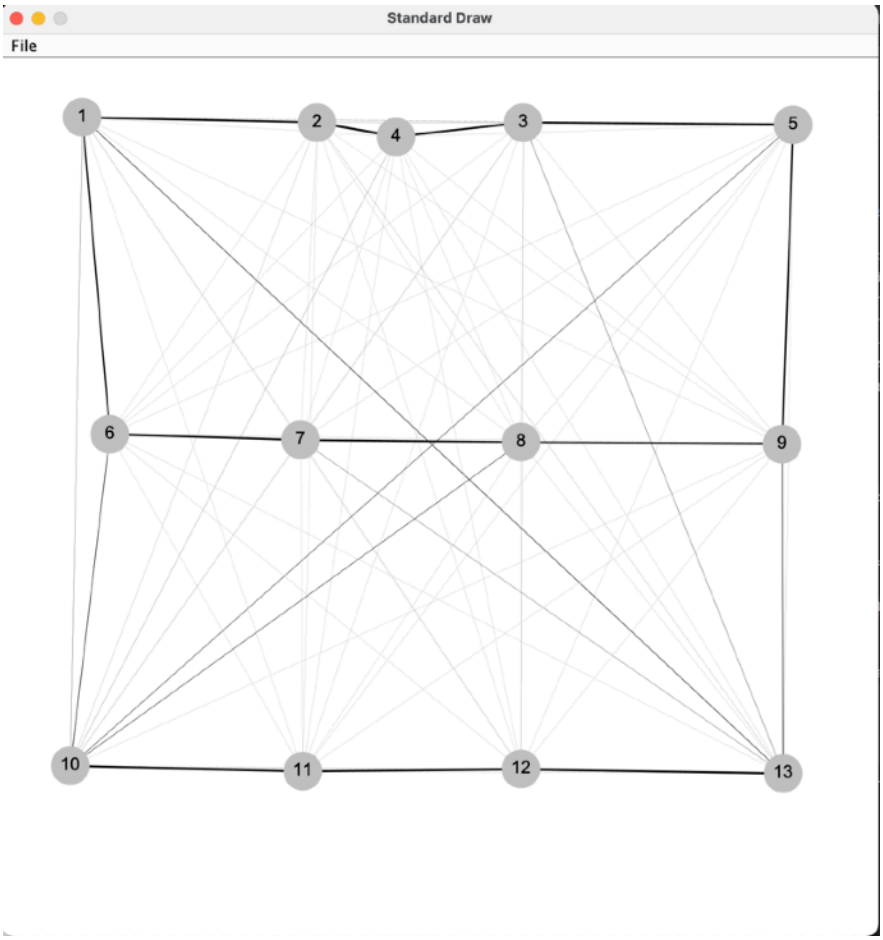
```java
// Constants to be used in ACO algorithm
final int iterationCount = 100; // Count of iterations
final int antCount = 50; // Size of colonies to be iterated over
final double degradationFactor = 0.8; // Evaporation rate
final double ALPHA = 1.4; // Alpha constant
final double BETA = 3.2; // Beta constant
final double initialPheromone = 0.001; // Initial pheromone on roads
final double Q = 0.0001; // Q constant
```

Input3

Shortest route found by both the bruteForce and antColony methods are the same for this approach.



The pheromone density map for antColony:



The outputs of said approaches are below. As seen one more time, antColony performs even faster compared to bruteForce than last time.

```
Method: Brute-Force Method
Shortest Distance: 3.80292
Shortest Path: [1, 2, 4, 3, 5, 9, 8, 13, 12, 11, 10, 7, 6, 1]
Time it takes to find the shortest path: 597.85 seconds.
```

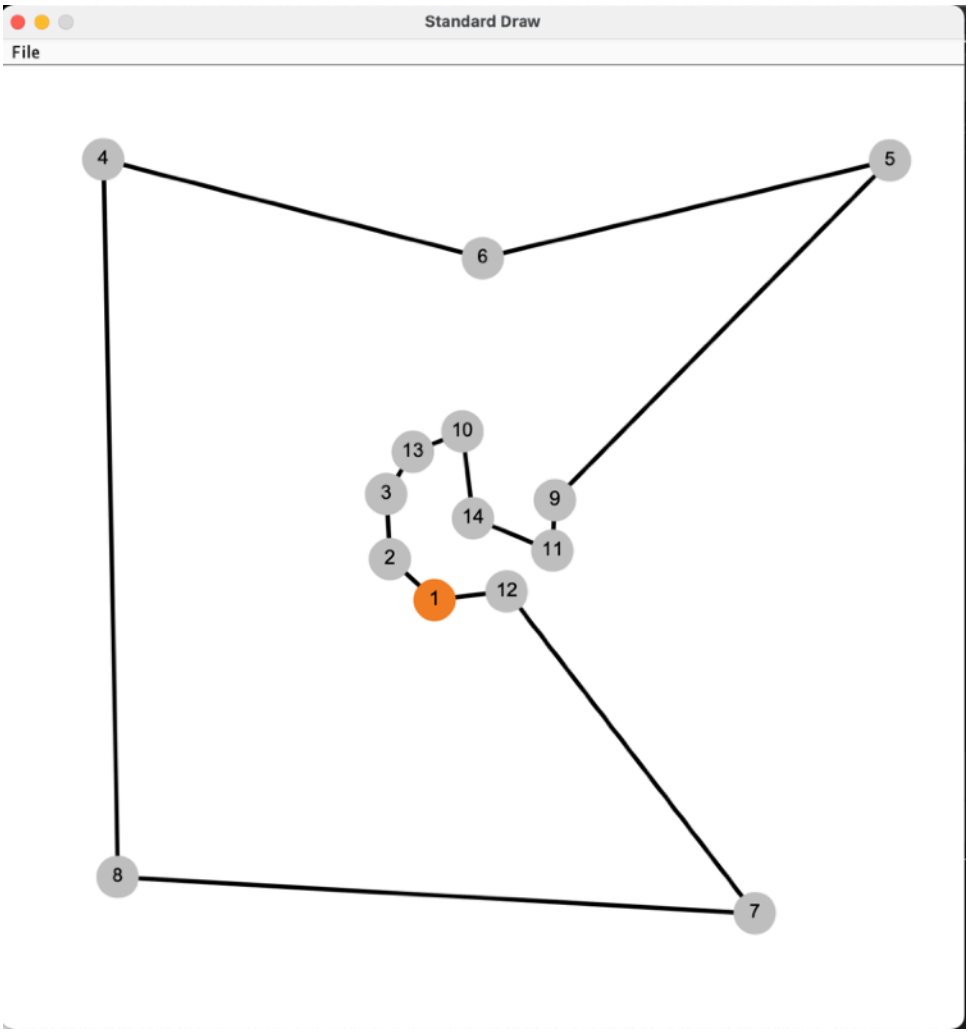
```
Method: Ant-Colony Method
Shortest Distance: 3.80292
Shortest Path: [1, 6, 7, 10, 11, 12, 13, 8, 9, 5, 3, 4, 2, 1]
Time it takes to find the shortest path: 0.18 seconds.
```

The ‘best’ parameters for antColony differ from the suggested parameters here as-well, as the suggested parameters can give a worse result most often.

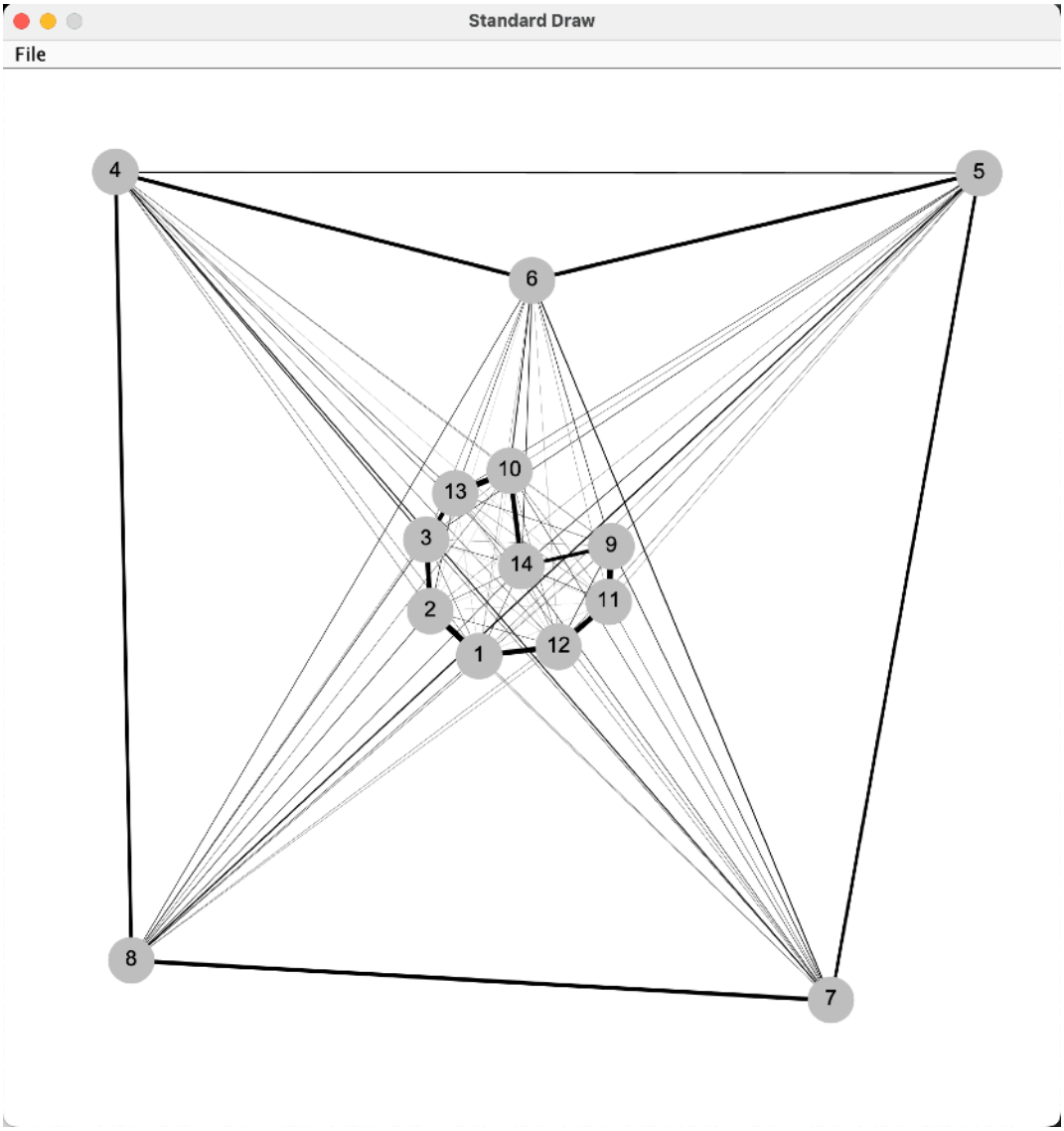
```java
// Constants to be used in ACO algorithm
final int iterationCount = 100; // Count of iterations
final int antCount = 50; // Size of colonies to be iterated over
final double degradationFactor = 0.7; // Evaporation rate
final double ALPHA = 2; // Alpha constant
final double BETA = 4; // Beta constant
final double initialPheromone = 0.001; // Initial pheromone on roads
final double Q = 0.0001; // Q constant
```

Input4

Shortest route found by both the bruteForce and antColony methods are the same for this approach.



The pheromone density map for antColony:



The outputs of said approaches are below. As seen, bruteForce takes a whopping 9000+ seconds of working time while antColony finishes its work in about a second.

```
Method: Brute-Force Method
Shortest Distance: 3.71091
Shortest Path: [1, 2, 3, 13, 10, 14, 11, 9, 5, 6, 4, 8, 7, 12, 1]
Time it takes to find the shortest path: 9505.40 seconds.
```

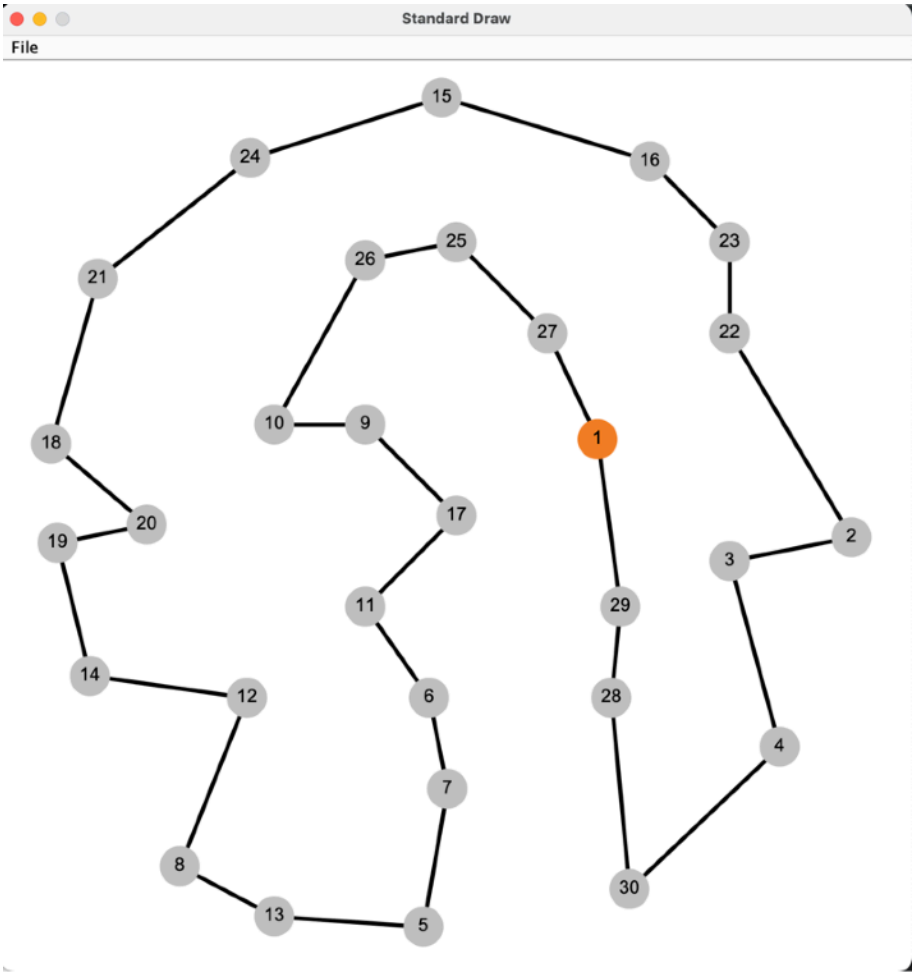
```
Method: Ant-Colony Method
Shortest Distance: 3.71091
Shortest Path: [1, 2, 3, 13, 10, 14, 11, 9, 5, 6, 4, 8, 7, 12, 1]
Time it takes to find the shortest path: 1.05 seconds.
```

The ‘best’ parameters for antColony differ from the suggested parameters here as-well, as the suggested parameters can give a worse result nearly always. We also need more iterations and ants since the input is bigger.

```
// Constants to be used in ACO algorithm
final int iterationCount = 500; // Count of iterations
final int antCount = 75; // Size of colonies to be iterated over
final double degradationFactor = 0.8; // Evaporation rate
final double ALPHA = 1.2; // Alpha constant
final double BETA = 2; // Beta constant
final double initialPheromone = 0.001; // Initial pheromone on roads
final double Q = 0.0001; // Q constant
```

Input5

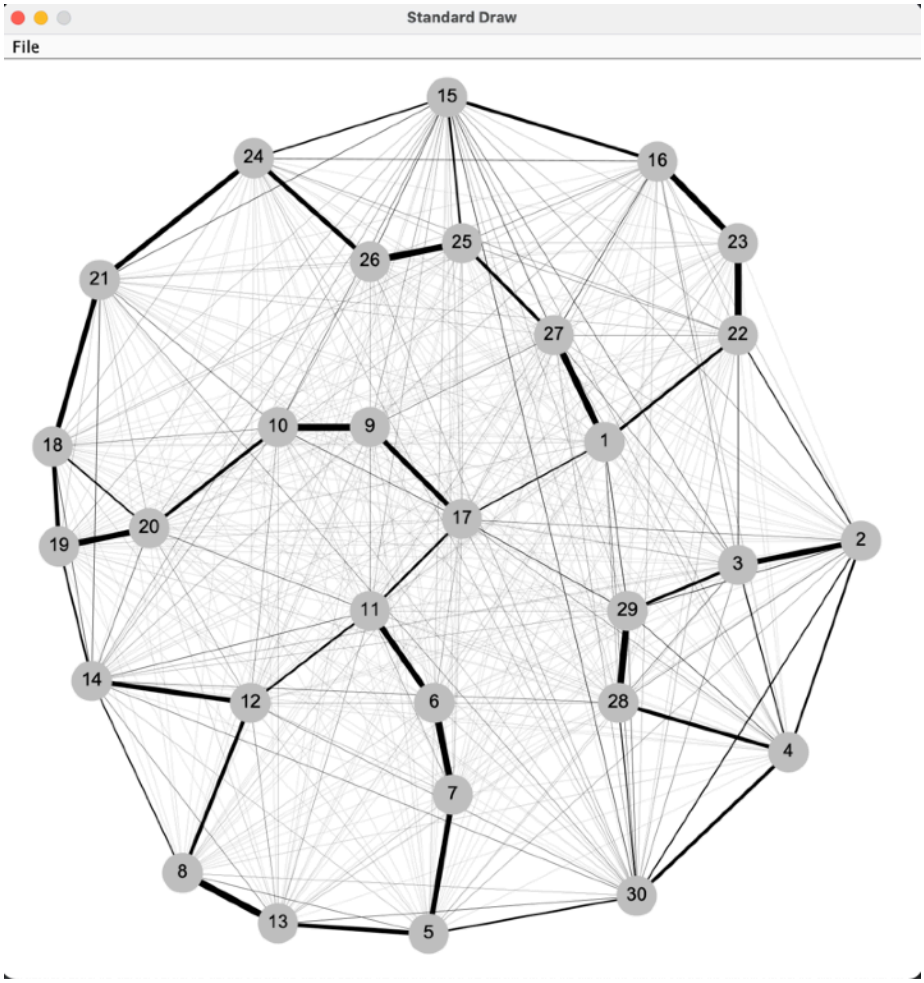
For this output, it would have taken millennia to even check one-tenth of the permutations. So, the shortest-for-certain path cannot be found but guessed. The shortest path that antColony can find needs to be relied on.



The path output:

```
Method: Ant-Colony Method
Shortest Distance: 4.79942
Shortest Path: [1, 27, 25, 26, 10, 9, 17, 11, 6, 7, 5, 13, 8, 12, 14, 19, 20, 18, 21, 24, 15, 16, 23, 22, 2, 3, 4, 30, 28, 29, 1]
Time it takes to find the shortest path: 3.84 seconds.
```

The pheromone densities look like these (Note that the densities are magnified in this output because degradation factor is 0.6, meaning that the pheromone levels get really low after some point):



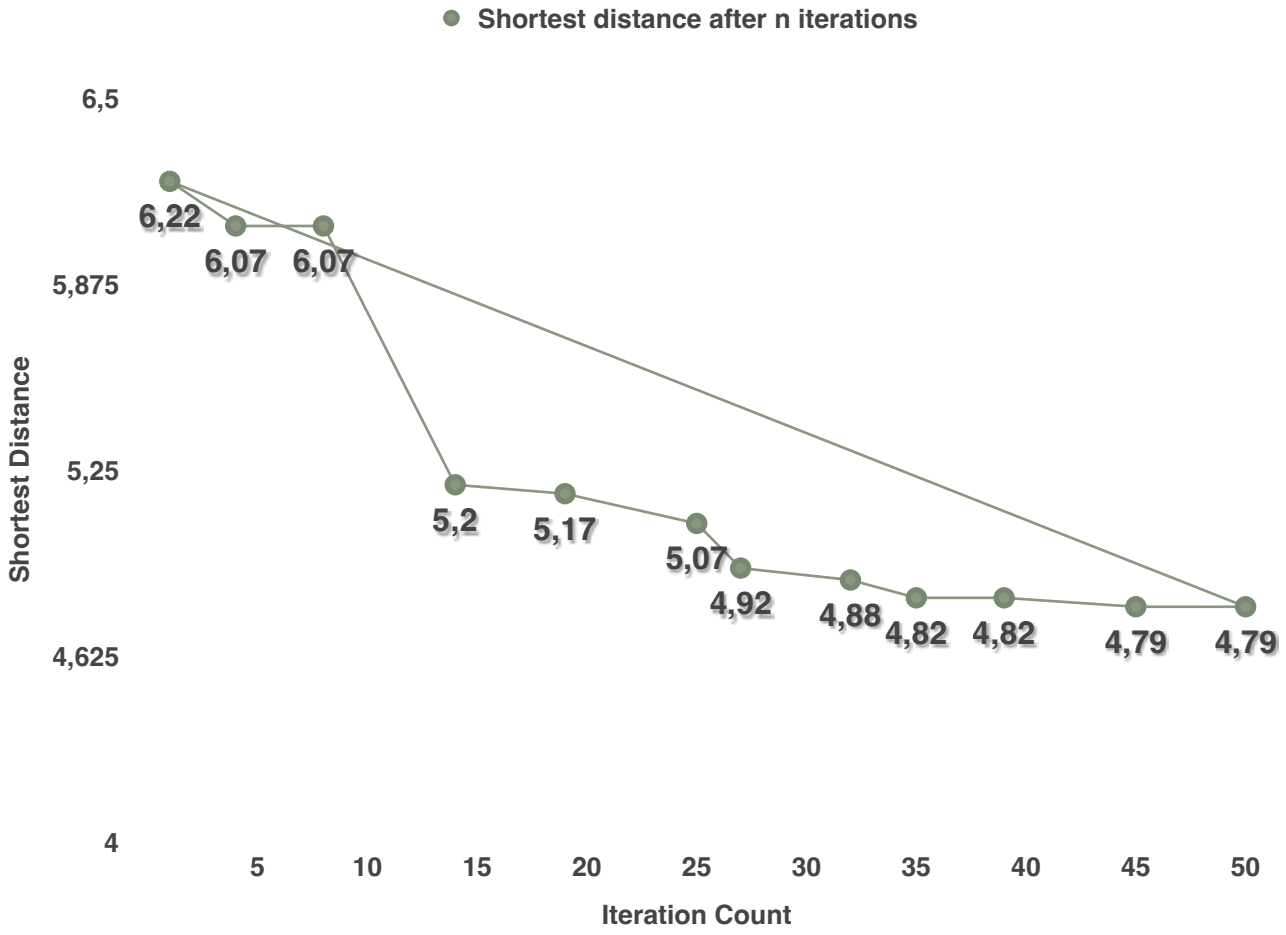
The parameters used for this input can also be considered the best overall set of parameters for all inputs. They seem to show the best results when tested.

```
// Constants to be used in ACO algorithm
final int iterationCount = 200; // Count of iterations
final int antCount = 100; // Size of colonies to be iterated over
final double degradationFactor = 0.6; // Evaporation rate
final double ALPHA = 0.8; // Alpha constant
final double BETA = 8; // Beta constant
final double initialPheromone = 0.001; // Initial pheromone on roads
final double Q = 0.0001; // Q constant
```

Section 2

Comparing the two algorithms

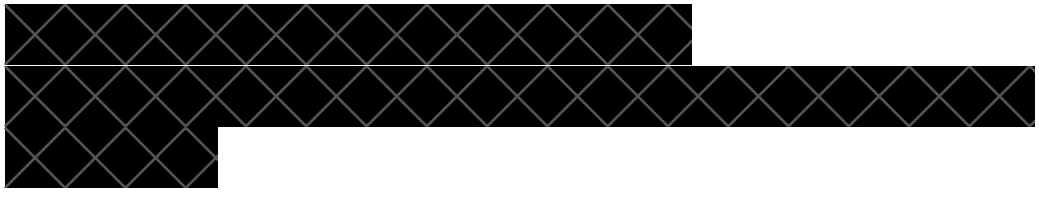
INPUT FILE	NUMBER OF HOUSES [redacted]	Brute-Force Time(SECONDS)	Ant Colony Time(SECONDS)	Speed Up Factor
INPUT1	11	2.91 seconds (distance: 1.79529)	0.13 seconds (distance: 1.79529)	22 times faster
INPUT2	12	41.95 seconds (distance: 2.93588)	0.17 seconds (distance: 2.93588)	247 times faster
INPUT3	13	597.85 seconds (distance: 3.80292)	0.18 seconds (distance: 3.80292)	3321 times faster
INPUT4	14	9505.40 seconds (distance: 3.71091)	1.05 seconds (distance: 3.71091)	9053 times faster
INPUT5	30	Too long to compute (presumably 9*10^11 million years)	3.84 seconds (distance: 4.79942)	Since brute-force is not computable, infinite times faster



References for my algorithms:

Brute-Force: https://en.wikipedia.org/wiki/Heap's_algorithm

Ant Colony: https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms for understanding the algorithm



Conclusion

Advantages / Disadvantages of ACO

We have seen that compared to brute-force, ant colony optimization is a much better approach in nearly every case. However, this superiority may not hold when all we want is nothing but accuracy.

Advantages of ACO:

1. Much faster and robust compared to most answers to the Traveling Salesman Problem.
2. Easy to understand, inspired by nature, and fairly easy to optimize using hyper-parameters.
3. One of the most easily implemented optimizations in the Traveling Salesman Problem.
4. Suitable for an Object Oriented Approach when implementing, for easy-to-grasp implementation.

Disadvantages of ACO:

1. When working with larger and larger inputs, the shortest path diverges from the real shortest path, making it inaccurate even with good hyper-parameters.
2. May not give the best answer every time even with the same hyper-parameters because it is a heuristic approach.
3. Nearly impossible to understand the mathematical logic to prove it gives the best result in a case.

In the end, all that matters is where and how you want to implement this algorithm. If it is a matter of time and space, this method is preferred. Otherwise, you may need to resort to finding more accurate methods.