## How the Prim algorithm works:

The prim algorithm begins before it's initial call during the createGraph() method. Within this method, the system reads a file, and declares the total number of vertices to global integer numVertex. Next, global integer array adjacency, is declared to be a symmetric n by n matrix of size numVertex. Following this, the prim algorithm is called, and parameter n is passed. Parameter n is of type integer and it represents the first vertex to be checked. Two variables of type integer are then created: min and vnear. Integer vnear is then initialized to equal the parameter n. Integer min is not initialized at this moment. Next, two one dimensional arrays of type integer are created; distance and nearest. Both of these arrays are then declared to be of size numVertex. Upon this declaration, both integer arrays: distance and nearest, are initialized within a for loop, whose position is declared by integer i. Integer array distance at position i, is initialized to be the value of the adjacency array at parameter n, as well as the position i. Integer array nearest at position i is then initialized to be the parameter n. Upon exit, the algorithm then begins a process contained in two nested loops. The outer loop parses through one less than the number of vertices, starting at the position given by parameter n, the loops's current position is given by integer i. Before the inner loop is declared, integer min is initialized to be the constant: INFINITY. The inner loop begins at position 1, and continues until the loops current position, given by integer j, has parsed through each value leading up to the number of vertices. Within the inner loop, each vertex is checked to see if it's edge is closest to the current vertex. If this returns true, integer min is set to the value of array distance at position j. Integer vnear is then set to position j. Upon exit of the inner for loop, array distance at position vnear is set to -1. This done such that vnear does not get re-added to the minimum spanning tree. After this, the system updates arrays distance and nearest to reflect the addition of vertex with subscript vnear to the minimum spanning tree. This is done through the use of a for loop, whose position (starting at 1) is noted by integer k, and ends when k has parsed through all the number of vertices. For every parse of this loop, the system checks to see if the value of adjacency at position k, position vnear, is less than the value at array distance at position k. If this returns true, array distance at position k is set to be the value at adjacency at position k, vnear. Array nearest at position k is then set to be the value of vnear. This for loop exits, and the outer for loop whose position is held by integer i increments by 1.

## Prim Complexity:

The prim time complexity is given by the function: $2(n-1)(n-1) \in \Theta(n^2)$
Where n is equal to the number of vertices. This is given because there are two main loops which the algorithm parses through. Each of these loops repeat a set of functions at least once, however only up to n-1 times.

## Examples of minimum spanning trees:

Two examples of minimum spanning trees that are of interest to me are:

- Clustering Analysis - an example being clustering points in the plane for supervised machine learning.
- Image registration and segmentation - an example being facial recognition using unsupervised machine learning.