1.
    a. Using the command (-g 2 -h 1 -n 6400 -s 0 -d 6399), so having a vertices value(N) of 6400, the program was able to run Dijkstra's algorithm with the worst case scenario in just over 1 second. 1.0329 to be exact. Using the command (-g 2 -h 1 -n 3200 -s 0 -d 3199) or N/2 number of vertices, the program was able to run the algorithm in about ¼ of the original time, or .2617 seconds to be exact. This proves that this performance has a complexity class of O(n^2) since splitting the vertices in half gave ¼ the time or (½)^2 = O(n^2).
    b. repeating the same steps but using graph 3 gave us a time of about 1.2885 seconds when using N to be 6400 and a time of about .314679 seconds when splitting N in half. This shows that this performance has a complexity class of O(n^2) because it is slightly slower than the worst case scenario given above but still takes about ¼ of the time when the number of vertices is split in half.
2. Using the network diameter option, and having 80 vertices we get a total time for it to all run of about 1.15096. the implementation is O(n^3) as it is dependent on every vertices to run through the algorithm N times. The time also significantly goes up. it is exponential.
3. -a 7

Seed	10	5	13	18	22	12424	5000	762	4422	1344
connect?	yes	no	no	no	no	no	no	no	no	no

-a 20

Seed	10	5	13	18	22	12424	5000	762	4422	1344
connect?	yes	no	yes	yes	yes	yes	yes	yes	yes	yes

4.

a.

```
Starting program: /home/huzefa/Dropbox/ECE_2
Seed: 1234567
cost: 185.000000 Path:0<-3<-8<-14
cost: 187.000000 Path:0<-4<-9<-14
cost: 187.000000 Path:0<-2<-7<-14
cost: 192.000000 Path:0<-1<-5<-10<-14
cost: 196.000000 Path:0<-6<-14
cost: 199.000000 Path:0<-5<-14
cost: 210.000000 Path:0<-7<-11<-14
cost: 226.000000 Path:0<-8<-12<-14
cost: 246.000000 Path:0<-9<-13<-14
cost: 270.000000 Path:0<-14
cost: 372.000000 Path:0<-10<-4<-14
cost: 438.000000 Path:0<-11<-6<-3<-14
cost: 506.000000 Path:0<-12<-6<-2<-14
cost: 580.000000 Path:0<-13<-7<-3<-1<-14
No Path exists from vertex 14 to vertex 0
```

run with command (-g 2 –h 3 –n 15 –s 14 –d 0)
For graph type 2, the number of paths is always one less than the number of vertices

b.

Trials	Vert. Average	Vert. Max	Vert. Min	Paths
R = 10	9	20	0	7
R = 20	18	34	3	10
R = 50	45	66	10	27
R = 100	85	126	23	59

As the R value went up or the adjacent nodes, the number of paths increased. however the lengths of the all the paths.

Test Plan

Unit Driver 0 shows a simple addedge implementation and then print out the resulting graph.

```
if (UnitNumber == 0)
{   go = 1;

        int V = 7;
        Num_Vertices = V;
        graph = newGraph(V);

        addEdge(graph, 0, 2, 10);
        addEdge(graph, 0, 1, 5);
        addEdge(graph, 0, 5, 4);
        addEdge(graph, 1, 0, 2);
        addEdge(graph, 1, 3, 7);
        addEdge(graph, 1, 5, 6);
        addEdge(graph, 2, 6, 4);
        addEdge(graph, 3, 4, 8);


        graph_debug_print(graph);
```

```
START OF GRAPH

Vertex 0
head-> 5 -> 1 -> 2

Vertex 1
head-> 5 -> 3 -> 0

Vertex 2
head-> 6

Vertex 3
head-> 4

Vertex 4
head

Vertex 5
head

Vertex 6
head
```

This correctly show the relation and adjacent vertices pairs as they were added in the graph and the print out.

Unit Driver 1 goes a little bit further because it creates a graph with vertices and then removes some of the edges.

```
if(UnitNumber == 1){
    go = 1;
    int V = 5;
    Num_Vertices = V;
    graph = newGraph(V);

    addEdge(graph, 0, 2, 10);
    addEdge(graph, 0, 1, 5);
    addEdge(graph, 0, 4, 4);
    addEdge(graph, 4, 3, 2);
    addEdge(graph, 1, 2, 2);
    addEdge(graph, 1, 0, 2);
    addEdge(graph, 3, 0, 2);
    addEdge(graph, 2, 4, 2);
    graph_debug_print(graph);
    graph_edge_remove(graph, 0, 1);
    graph_debug_print(graph);
    graph_edge_remove(graph, 0, 2);
    graph_edge_remove(graph, 0, 4);
    graph_debug_print(graph);
```

```
START OF GRAPH

Vertex 0
head-> 4 -> 1 -> 2

Vertex 1
head-> 0 -> 2

Vertex 2
head-> 4

Vertex 3
head-> 0

Vertex 4
head-> 3
```

```
START OF GRAPH

Vertex 0
head-> 4 -> 2

Vertex 1
head-> 0 -> 2

Vertex 2
head-> 4

Vertex 3
head-> 0

Vertex 4
head-> 3
```

```
START OF GRAPH

Vertex 0
head

Vertex 1
head-> 0 -> 2

Vertex 2
head-> 4

Vertex 3
head-> 0

Vertex 4
head-> 3
```

Unit Driver one shows how each edge was added and then prints the resulting graph. One edge is removed and from the second terminal picture you can see that the edge between vertices 0 and 1 was removed. The third terminal pictures shows that all edges from 0 were removed. This add edge and remove edge works.