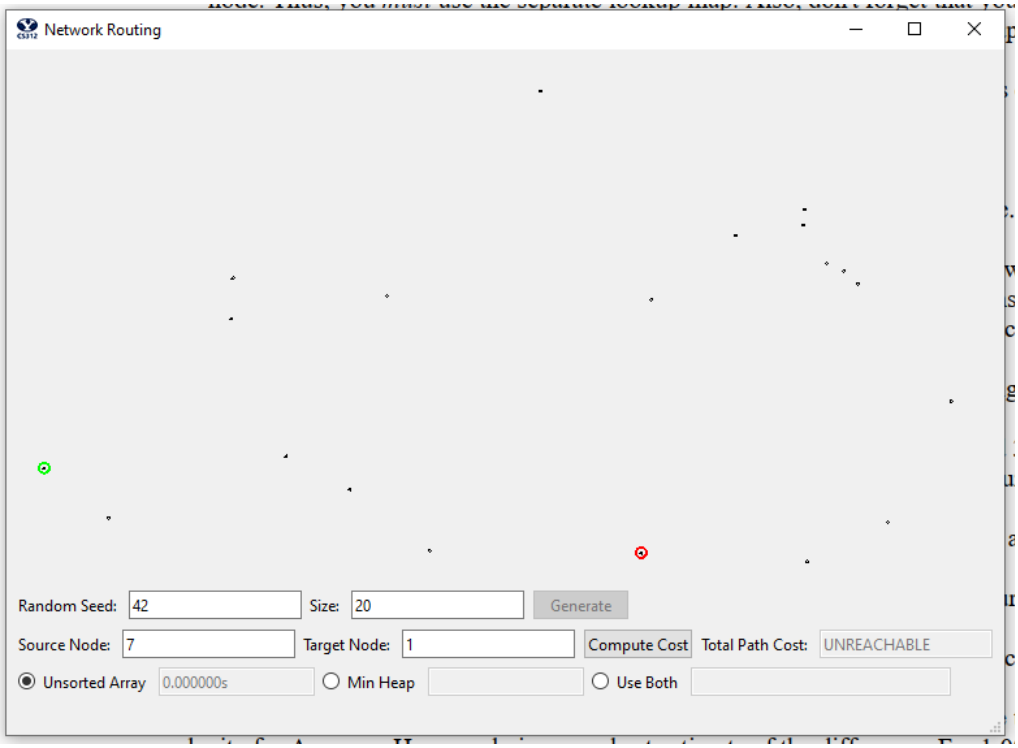
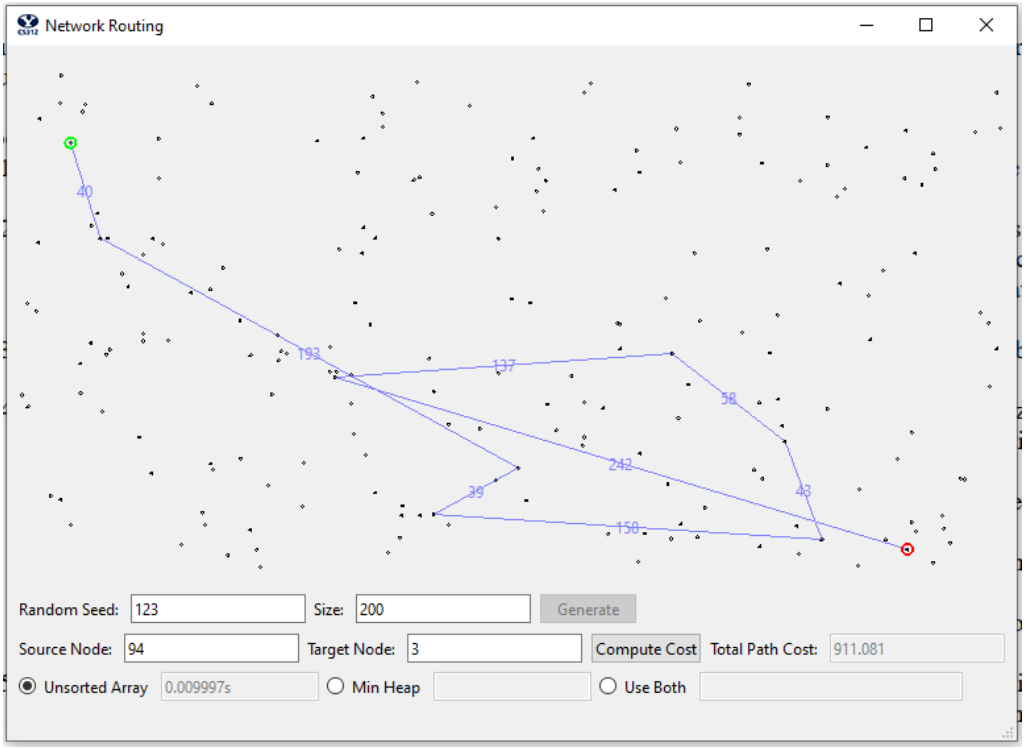
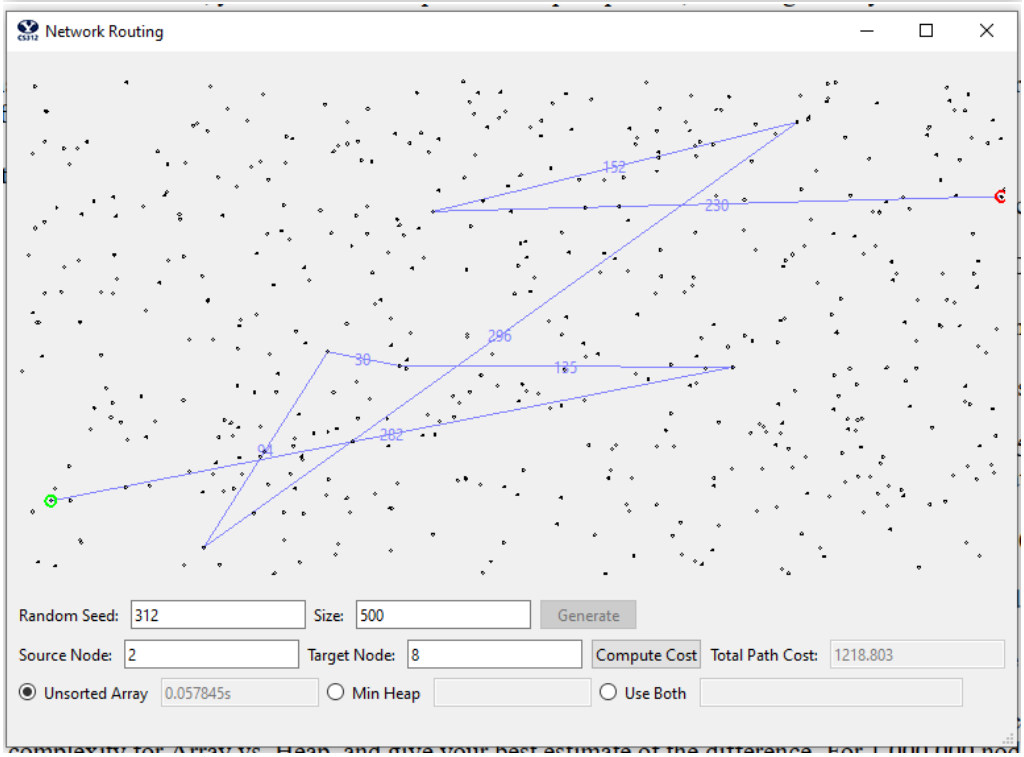
```
Function Dijkstra:
Input: network and points
Output: map of nodes with distance and previous node
For each node (v)
        Insert()
Set source distance to 0
While size(queue) > 0 (v)
        currentNode = deletemin()
        for neighbors of currentNode (max v per iteration, total e over all iteration)
                weight = currentNode weight + edge weight
                if weight < endNode weight
                        decreasekey(endnode, weight)
                        updatePreviousNode(endnode, currentNode)
Heap
DecreaseKey()
Input: index, weight
Output: none
Change weight of node at index
bubbleUp()

DeleteMin()
Input: none
Output: node with lowest weight
Swap first and last nodes
Pop and store new last node
siftDown()
return stored node

Insert()
Input: node and weight
Output: none
Add new node to end
bubbleUp()

Array
DecreaseKey()
Imput: node index and weight
Output:none
Update weight of node at index

DeleteMin()
Input: none
Output: node with lowest weight
Search for lowest weight and remove from array, return it
```

Insert()
Input: node and weight
Output: none
Add to end of array




Network Routing
40
193
137
58
242
39
43
158
Random Seed: 123
Size: 200
Generate
Source Node: 94
Target Node: 3
Compute Cost
Total Path Cost: 911.081
Unsorted Array
0.009997s
Min Heap
Use Both


Network Routing
152
230
286
30
135
94
282
Random Seed: 312
Size: 500
Generate
Source Node: 2
Target Node: 8
Compute Cost
Total Path Cost: 1218.803
Unsorted Array
0.057845s
Min Heap
Use Both