

For a heap, insertion is O(log(v)), since it is simply appending to the queue(which is O(1)) and then bubbling up, which is log(v) time.

For the unsorted array, insertion is O(1), as it would simply append to the queue; in this case I just returned a copy of the array in make\_array, which is O(v), but there would be v insertions anyway, so it is the same time complexity.

Bubble\_up is log(v) because it will at worst iterate a number of times equal to the height of the tree.

Heap\_delete\_min is log(v) because pop() is an O(1) operation and sift\_down is log(v) (for the same reason bubble\_up is log(v))

Array\_delete\_min is O(n) because it has to iterate though the list to find the correct index, then pop(index) is O(n)

The array implementation does not have a decreaseKey function, as it would be pointless. Since the array is unsorted, there is no need to notify anything but the node itself when a distance is updated.

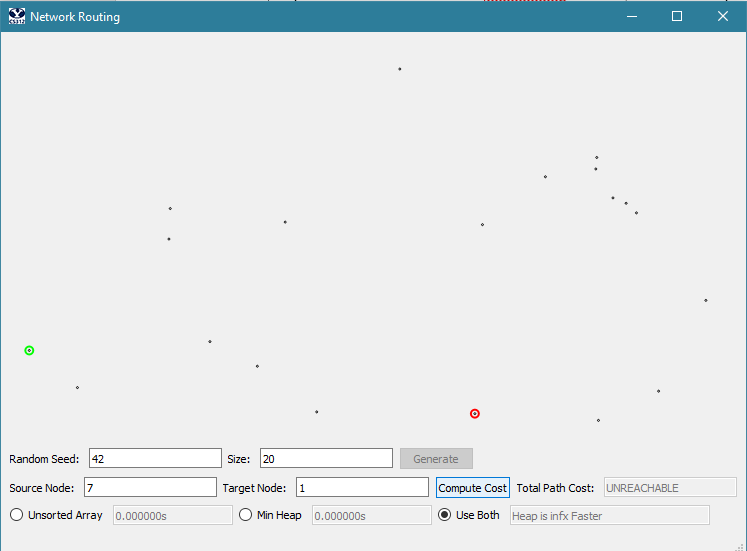
Decrease\_key is also log(v) since all it does is call bubble\_up; the node value in the queue has already been updated on the line previous to the call to decrease\_key

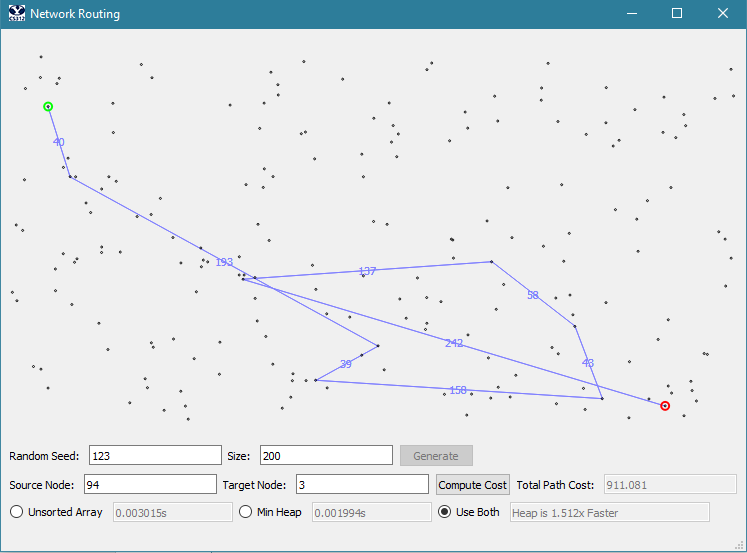
The space complexity of the unsorted array implementation is simply O(v), as it only ever keeps track of the single array

The space complexity of the heap implementation is O(2v) = O(v), as it keeps track of the array as well as a dictionary of node\_indices which is size v.

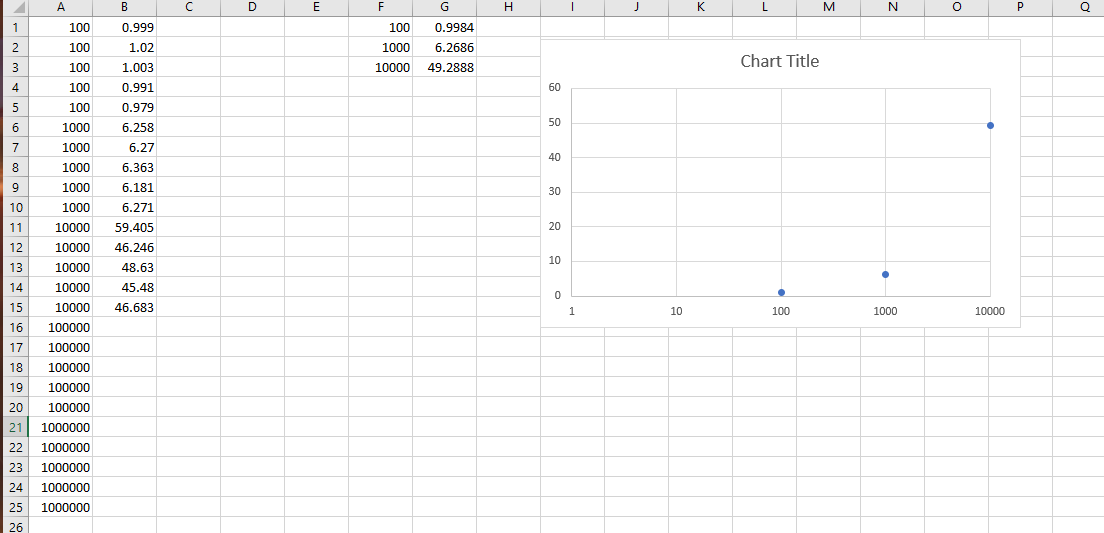
The overall time complexity for the unsorted array implementation is O(v^2), as it will go through each node (O(v)) and for each node, call delete\_min, which adds another factor of v.

The overall time complexity for the heap implementation is O(vlog(v)), as it will go through each node (again, O(v)) and for each node, call each of insert, delete\_min, and decrease\_key, adding up to O(log(V) asymptotically, resulting in O(vlog(v)) overall.









Unfortunately I ran out of time to compare array to heap implementation on the 100000 tests, but my heap seemed to get about another 10x faster the more nodes it had to go through, so I would estimate that my heap would be about 5000x faster than the array implementation for 1000000 nodes.