Big Oh

Dijkstra

*O*((*n* + *m*) log *n*)

Using the priority queue implementation, we have to implement a way to change the keys as we change the weights. Since we know where to place it will take O(log n) time to do key updates. We then have to go through each vertex and its edges which adds the O(n+m). In the end its *O*((*n* + *m*) log *n*).

MST

*O*((*n* + *m*) log *n*)

Using the priority queue implementation, we have to extract each vertex in each iteration which takes O(log n) time. In addition, we can update each cost value in O(log n) which is considered at most once for each edge. The other stages of the algorithm are O(1) time so this algorithm ends out being O((n+m) log n).

BFS

O(m (n+m))

With the adjacency list, we have to iterate through the edge list for the current vertex. This takes O(m) times (m is the amount of edges connected to the vertex). This is also a condition for a while loops. Inside the loop we have to iterate through the edge list again to fine the smallest edge. That also takes O(m) amount of time. Next, we iterate through the vertex list which takes O(n) (n is the amount of vertices). Outside of the loop we can ignore the individual loop that takes O(m) and O(n) respectively since they are smaller than the run time of the previous loop. The run time for this algorithm is O(m(n+m)).

DFS

O(n + m)

Among the methods used in our software is a Depth-First Search. In this method, each vertex n of our graph is initially labeled as unexplored, which can be identified as n labels. Each edge m of the graph is also labeled as unexplored, identified as m labels. As the function loops, each vertex will be labeled again as visited, another n labels. Each edge is labeled as discovery if used in our function, and then labeled as back if not used, another m labels. Therefore, the operation can be visualized as n + n + m + m operations. This is simplified to an O(n + m) running time for DFS.