Team Slam Jam Big-Oh Analysis

void LoginWindow::on_loginButton_clicked()

The Big Oh of this function is O(1) because it queries our SQL database for username and password information, and performs one calculation for each username and password combination in the database. It compares each of these combinations with the username and password given by the user in order to find a match. The function always performs the same number of calculations regardless of user input, making this function O(1)

std::vector<Souvenir> querySouvenirs(QString teamName)

The Big Oh of this function is O(N) because it consists of three separate For Loops, each of which run at O(n). The first two For Loops retrieve all souvenir names and souvenir prices from the SQL database and push them to a vector, and each perform N calculations, where N is the number of souvenirs in the database. The third For Loop constructs a vector of Souvenir class objects using the data from the name and price vectors. It performs 2*N calculations: N calculations to construct Souvenir objects, and N calculations to push them to a Souvenir vector. Because each for loop performs N calculations and none of them are nested, the overall efficiency is O(N)

std::vector<Edge<T>> BFS(T start) *(In MatrixGraph.h)*

The Big Oh of this function is O($N^2 logN$), where N is the number of vertices in our graph. First, we search the graph for the starting vertex, which takes N calculations. Then, we enter a while loop which loops N times. Inside this loop, we search for the index location of the

next vertex that will be visited, which takes N calculations. Then we sort each of this vertex's neighboring vertices using a sorting algorithm which takes Nlog(N) time. Then each of these vertices is added to a vector to be visited, which takes N comparisons. The overall efficiency of the algorithm is O($N^2 logN$) because we perform a sort which takes NlogN time inside of a loop which occurs N times.

std::vector<Edge<T>> MatrixGraph<T>::kruskalMST()

The Big Oh of this function is O($NlogN$), where N is the number of vertices in our graph. First, it sorts the edges by weight using a comparison sort in $O(N \log N)$ time; Then, the algorithm performs an O(N) loop where it looks at the edge with the lowest weight, and uses a disjoint-set data structure to determine if that edge forms a cycle with other edge currently in the MST. This takes O(N) an additional operations, as in each iteration we connect a vertex to the spanning tree, then perform two 'find' operations and possibly one union for each edge, each of which take N calculations. Thus the total time is $O(N \log N)$.

void MatrixGraph<T>::addEdge(T start, T end, double weight)

The Big Oh of this function is O(N), where N is the number of vertices in the graph. First, the function iterates through the current vertices in the graph and determines the id of the vertices in the edge, which takes 2*N calculations. Then, the edge is placed into the adjacency matrix, which is O(1). Finally, the edge is inserted into the graph as a bidirectional edge going to and from each vertex which is O(1).