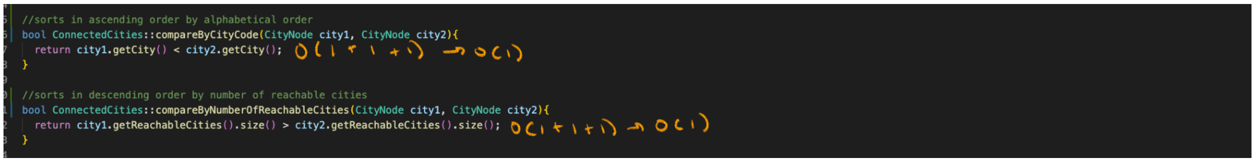
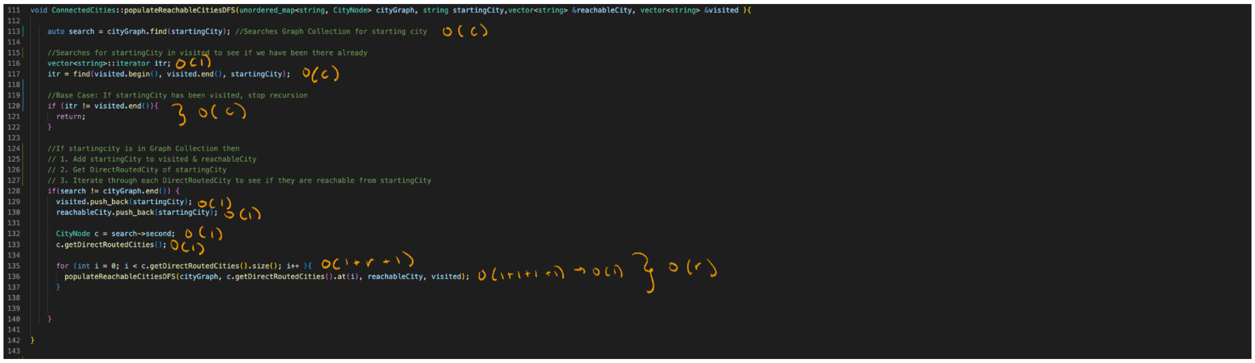
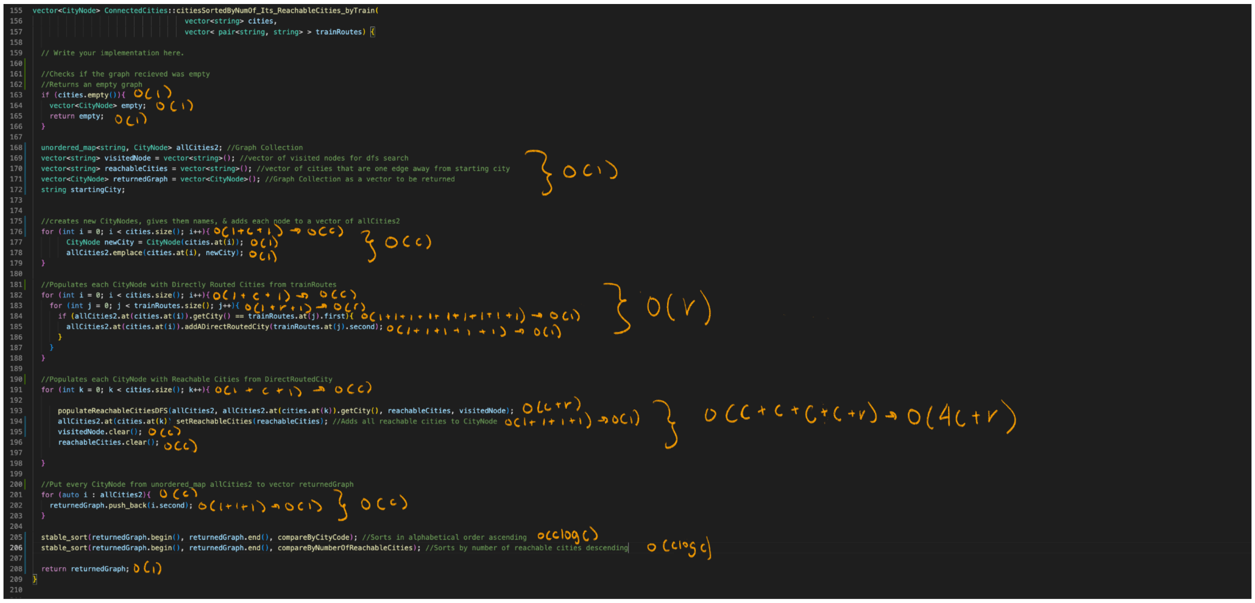


```
//sorts in ascending order by alphabetical order
bool ConnectedCities::compareByCityCode(CityNode city1, CityNode city2){
  return city1.getCity() < city2.getCity();  O(1 + 1 + 1) → O(1)
}

//sorts in descending order by number of reachable cities
bool ConnectedCities::compareByNumberOfReachableCities(CityNode city1, CityNode city2){
  return city1.getReachableCities().size() > city2.getReachableCities().size();  O(1 + 1 + 1) → O(1)
}
```



```
void ConnectedCities::populateReachableCitiesDFS(unordered_map<string, CityNode> cityGraph, string startingCity,vector<string> &reachableCity, vector<string> &visited ){

    auto search = cityGraph.find(startingCity); //Searches Graph Collection for starting city   O(c)

    //Searches for startingCity in visited to see if we have been there already
    vector<string>::iterator itr; O(1)
    itr = find(visited.begin(), visited.end(), startingCity);   O(c)

    //Base Case: If startingCity has been visited, stop recursion
    if (itr != visited.end()){      } O(c)
      return;
    }

    //If startingcity is in Graph Collection then
    // 1. Add startingCity to visited & reachableCity
    // 2. Get DirectRoutedCity of startingCity
    // 3. Iterate through each DirectRoutedCity to see if they are reachable from startingCity
    if(search != cityGraph.end()) {
      visited.push_back(startingCity); O(1)
      reachableCity.push_back(startingCity); O(1)

      CityNode c = search->second;  O(1)
      c.getDirectRoutedCities(); O(1)

      for (int i = 0; i < c.getDirectRoutedCities().size(); i++ ){  O(1 + r + 1)
        populateReachableCitiesDFS(cityGraph, c.getDirectRoutedCities().at(i), reachableCity, visited);  O(1+1+1+1) → O(1)  } O(r)
      }


    }

}
```



```
vector<CityNode> ConnectedCities::citiesSortedByNumOf_Its_ReachableCities_byTrain(
                                    vector<string> cities,
                                    vector< pair<string, string> > trainRoutes) {

  // Write your implementation here.

  //Checks if the graph recieved was empty
  //Returns an empty graph
  if (cities.empty()){  O(1)
    vector<CityNode> empty;  O(1)
    return empty;  O(1)
  }

  unordered_map<string, CityNode> allCities2; //Graph Collection
  vector<string> visitedNode = vector<string>(); //vector of visited nodes for dfs search
  vector<string> reachableCities = vector<string>(); //vector of cities that are one edge away from starting city   } O(1)
  vector<CityNode> returnedGraph = vector<CityNode>(); //Graph Collection as a vector to be returned
  string startingCity;


  //creates new CityNodes, gives them names, & adds each node to a vector of allCities2
  for (int i = 0; i < cities.size(); i++){ O(1 + c + 1) → O(c)
      CityNode newCity = CityNode(cities.at(i)); O(1)      } O(c)
      allCities2.emplace(cities.at(i), newCity); O(1)
  }

  //Populates each CityNode with Directly Routed Cities from trainRoutes
  for (int i = 0; i < cities.size(); i++){ O(1 + c + 1) → O(c)
    for (int j = 0; j < trainRoutes.size(); j++){ O(1 + r + 1) → O(r)
      if (allCities2.at(cities.at(i)).getCity() == trainRoutes.at(j).first){ O(1+1+1+1+1+1+1+1) → O(1)   } O(r)
        allCities2.at(cities.at(i)).addADirectRoutedCity(trainRoutes.at(j).second); O(1 + 1 + 1 + 1 + 1) → O(1)
      }
    }
  }

  //Populates each CityNode with Reachable Cities from DirectRoutedCity
  for (int k = 0; k < cities.size(); k++){ O(1 + c + 1) → O(c)

    populateReachableCitiesDFS(allCities2, allCities2.at(cities.at(k)).getCity(), reachableCities, visitedNode); O(c + r)
    allCities2.at(cities.at(k)).setReachableCities(reachableCities); //Adds all reachable cities to CityNode  O(1 + 1 + 1 + 1) → O(1)   } O(c + c + c + c + r) → O(4c + r)
    visitedNode.clear(); O(c)
    reachableCities.clear(); O(c)

  }

  //Put every CityNode from unordered_map allCities2 to vector returnedGraph
  for (auto i : allCities2){  O(c)
    returnedGraph.push_back(i.second); O(1 + 1 + 1) → O(1)  } O(c)
  }

  stable_sort(returnedGraph.begin(), returnedGraph.end(), compareByCityCode); //Sorts in alphabetical order ascending  O(clogc)
  stable_sort(returnedGraph.begin(), returnedGraph.end(), compareByNumberOfReachableCities); //Sorts by number of reachable cities descending   O(clogc)

  return returnedGraph; O(1)
}
```

$$O(1) + O(1) + O(1) + O(1) + O(c) + O(r) + O(4c + r) + O(c) + O(c\log c) + O(c\log c) = \boxed{O(4 + 6c + 2r + 2c\log c)}$$