**Let’s play Football**

**Project Number: 2**

**Big Oh Paragraph**

The data structure that our team – The Brogrammer – uses for this second Project is an artificial map. This project doesn’t use any databases such as MySQL, it uses a text file instead. There are distinguish text file for stadium information, item price information, revenue information, graph information. There are also other features that need to check for their big Oh.

The read from file method/functions (loadStadiumListFromFile(string filename), loadSouvenirs(string filename), loadDefaultGraph()) use a while loop to read line by line until it reach the end of the file (while (!inFile.eof())) Thus it **takes O(n) times** to read and write all information from each kind: Stadium, Item Price, Revenue, Souvenir.

1. The artificial map used in this project is a doubly linked list with the map functionalities:

* First, the map used the StadiumNode which contains a key, a value, and the next and previous pointers.
* The \*find(QString aKey) method searches the map for the key according to the parameter. Since this is doubly, the searching **will takes O(n) times**, which means the worst case scenarios will be the found node as the last node in the doubly linked list
* The \*put(QString aKey, Stadium &aValue) method adds the node to the doubly linked list. The adding is based on the helper function - \*insert(QString aKey, Stadium aValue) – This adding also **takes O(n) times** since it might have to add at the end of the list for the worst case.
* The remove(QString aKey) method removes the node from the doubly linked list. This removal use is based on the helper function - removeNode(QString aKey) – This removal will **takes O(n) times** as it probably have to traverse to the last node to delete the node.
* The \*begin() const method returns the address of the first node of the list. This method **takes O(1) time** which is at **constant time**.
* The \*end() const method returns the address of the last node of the list. Since it has to traverse to last node in order to get the address, it would **take O(n) times**

1. The Add/Modify Stadium Class:

* The loadDefaultSouvenirList() method reads the souvenir file and store the information souvenir. This **takes O(n) times** since it made a while to read all of the information
* There is no method for reading the stadium text file, but it was created inside other methods. The performance of reading the text **takes O(n) times** as there is a while loop to read those information

1. View Team/Stadium Class

* Whether viewing a specific team or the list of team, the performance **takes O(n) times**, since the stadium information is stored in the doubly linked list and in order to read it, one must traverse it from the beginning to the end.

1. Sorting Team/Stadium Class

* Before the list of Teams/Stadiums can be sorted. The whole list needs to be called, which would also **take O(n) times** as the software needs to read to traverse the entire list in order to get all of the list information.

1. Trip planner class

* This class implements Dijkstra's algorithm to get the shortest distance. Adjacent Vertexes are used for this class. Since two loop need to be made(inside the list of vertexes and inside one vertex), the performance will **take O() times**