Steamed Potatoes Big-Oh Analysis

1.

```
void Mainwindow::on_loginbutton_clicked()
{
      if(ui->UserLineEdit->text() == "admin" && ui->PWLineEdit->text() ==
      "potato")
      {
             //ADMIN PAGE
            QMessageBox::information(this, "Welcome, Admin!",
            "Now moving to Main Admin Screen...", QMessageBox::Ok,
            QMessageBox::NoButton);                                          1
            adminObj.show();                                                 1
      }
      else
               QMessageBox::critical(this, "Access Denied", "INCORRECT
               LOGIN", QMessageBox::Ok, QMessageBox::NoButton);              1

{
FUNCTION BIG-OH = 1 + 1 + 1 = O(1)
```

The two lines edits, UserLineEdit and PWLineEdit, read in two strings in constant time. The string that is valid for UserLineEdit is “admin” whereas the string that is valid for PWLineEdit is “potato”. Ultimately, the constant checks for these two valid strings. If either of the line edits are NULL or the value inputted in the line edits do not match, a QMessageBox Critical will appear. This will deny them access to the admin page. Conclusively, the Big-Oh of this function is 1 or constant. Regardless of user input, the program does not run inefficiently.

- O(1)

Overall the function **on_loginbutton_clicked**() has a time of

- O(1)

2.

```
void adminpage:: on_deleteSouvenirConfirmBtn_clicked()
{
    int souvCount = 0;                                                       1
    int afterSouvCount = 0;                                                  1

    QString nameCampus = ui->deletePageComboBox->currentText();              1
    QString nameSouv = ui->deletePageSouvenirName_line->text();              1

    QSqlQuery data;                                                          1
    QSqlQuery dataOne;                                                       1
    QSqlQuery dataTwo;                                                       1
    QSqlQuery qry;                                                           1

    souvCount = data.prepare("SELECT COUNT(souvCollege) FROM Souvenirs");    n
    data.exec();                                                             1

    if(data.next())                                                          1
    {
        souvCount = data.value(0).toInt();                                   1
    }
```

```
    qry.prepare("DELETE FROM Souvenirs WHERE souvCollege = '"+nameCampus+"'
AND souvTrad = '"+nameSouv+"'");                                          n
    qry.exec();                                                           1
    dataOne.exec();                                                       1

    afterSouvCount = dataTwo.prepare("SELECT COUNT(souvCollege) FROM
Souvenirs");                                                              n
    dataTwo.exec();                                                       1

    if(dataTwo.next())                                                    1
    {
        afterSouvCount = dataTwo.value(0).toInt();                        1
    }

    if(souvCount == afterSouvCount)
    {
        QMessageBox::about(this, "Error", "Value not found double check path
to database!");                                                           1
    }
    else
    {
        QMessageBox::about(this, "", "The item was deleted. Double check if
an error occured");                                                       1

        ui->stackedWidget->setCurrentWidget(ui->adminHomePage);           1
    }
}
FUNCTION BIG O = 1+1+1+1+1+1+1+1+n+1+1+n+1+1+n+1+1+1+1+1 = O(3n+18) = O(n)
```

Declarations and initializations of all local variables souvCount, afterSouvCount, nameCampus, nameSouv, data, dataOne, dataTwo, qry are done in constant time. The Qstrings nameCampus and nameSouv are intitalized through the current text in the lineEdits and combo boxes.

- O(1)

The SQL queries prepared using SELECT and DELETE were executed in O(n) because the worst case scenario is to search through the whole database of n rows to find specified query.

- O(n)

All the executions of the queries, .exec(), are in constant time because the executions return the rows executed of the query.

- O(1)

All the .next() functions are executed in constant time because .next() retrieves the next record in the result.

- O(1)

All the .value(0).toInt functions are executed in constant time because the functions retrieves the integer value of the result of the execution of the next record.

- O(1)

The line edit, deletePageSouvenirName_line, reads in a string in constant time. The combo box, deletePagecomboBox, has the user select an existing string also in constant time. The constant will check if the line edit does not contain values that are either an integer or NULL. If the constant remains true, then the QSqlQuery variable will execute and output a QMessageBox statement and perform the action on the database. If the constant discontinues, then it will output

a QMessageBox statement and halt any action on the database. Thus, the Big-Oh of this function is 1 or constant.

- O(1)

Setting the interface widget to the admin home page if the number of rows before and after the delete query were not equal is constant time because input size does not affect the execution of this statement

- O(1)

Overall, the function **on_deleteSouvenirConfirmBtn_clicked**() has a time of

- O(n)

3.

```
void studentpage::recursiveCollegeSort(QString currentCamp)
{
   sortedCampuses.enqueue(currentCamp);                                    1
   selectedCampuses.removeAll(currentCamp);                                1

   if(selectedCampuses.isEmpty())
   {
       return;                                                             1
   }
   else
   {
        QString* otherCamp = selectedCampuses.begin();                     1
        int currentIndex = 0;                                              1

        double leastDist = database.GetDistBtwn(currentCamp, *otherCamp);  1
        int leastIndex = 0;                                                1

        otherCamp++;                                                       1
        currentIndex++;                                                    1

        while(otherCamp != selectedCampuses.end())                         n
        {
        double currentDist = database.GetDistBtwn(currentCamp,
        *otherCamp);                                                       n^2

            if (currentDist < leastDist)
            {
                leastDist = currentDist;                                   n
                leastIndex = currentIndex;                                 n
            }
            else
            {
                otherCamp++;                                               n
                currentIndex++;                                            n
            }
        }

        QString nextCamp = selectedCampuses.at(leastIndex);                1
        recursiveCollegeSort(nextCamp);                    recursive runs = n
   }
}
```

FUNCTION BIG-OH = (1+1+1+1+1+1+1+1+1+n+n^2n+n+n+n+1) * n recursive runs = O(n^3)

This function creates the optimal college touring trip plan. For example, from the starting campus, the next campus will be the closest to the start, then the closest after that. Each recursive call finds the closest campus to the campus passed in. When the closest is found, the new campus is sorted and then used for the next call.
The enqueue() to sortedCampuses and removeAll() from selectedCampuses both run in constant time.

- O(1)

The base case of the recursive function, if(selectedCampuses.isEmpty()), and the return statement result in constant time.

- O(1)

In the recursive case, the initializations of otherCamp, currentIndex, leastDistance, and leastIndex all run in constant time.

- O(1)

The increments of otherCamp and current index run in constant time

- O(1)

The while loop that traverse through selectedCampuses searching for the closest campus results in the body of the loop running n times.

- O(n)

Within the body of the loop, reassignments of leastDist & leastIndex and the incrementing of otherCamp & currentIndex all have a constant runtime. However, these executions run n number of times.

- O(n)

Moreover, within the body of the loop, the initialization of current distance using the GetDistBtwn(~) results in O(n) time because the select query function has a Big-Oh of O(n) from the worst case searching through the all n rows of the database. However, these executions run n number of times.

- O(n^2)

Since the whole function runs recursively for each campus selected by the student, the function will recursively iterate n times, O(n). The single iteration of the whole function was O(n^2), but since the recursive call runs this function again n times. The overall recursive function will in O(n^3).

- O(n^3)

As a result, a single iteration having a Big-Oh of O(n^2) due to the body of the loop have a big-Oh of O(n) and running n times. Moreover, because the whole function iterates recursively n times, the final Big-Oh for the whole recursive function would result in a big O of O(n^3). Time complexity of function **recursiveCollegeSort**(QString currentCamp) is as follows:

Single iteration = O(n^2)
Whole recursive function = O(n^3)

```
4. void studentpage::calcTotalDist()
{
    tourDist = 0;                                                    1
    for(int i = 1; i < sortedCampuses.count(); i++)                  n
    {
        double currentDistBtwn = database.GetDistBtwn(sortedCampuses[i-1],
sortedCampuses[i]);                                                  n^2
```

```
        tourDist += currentDistBtwn;                                    n
    }
}
```
FUNCTION BIG-OH = n+n^2+1 = O(n^2)

Using the sorted campus tour plan, the total distance traveled is calculated.
The initialization of start, end, and tourDist at the beginning of the function result in constant time,

- O(1).

Because there is a single for loop that traverse the sortedCampuses vector using a counter i starting 1, the body of the loop will repeat n times,

- O(n).

Within the body of the loop, the database.GetDistBtween(~) function utilizes the select query function, which has a Big-Oh of O(n). In addition, the addition of tourDist and currentDistBtwn results in constant time, O(1). However, these executions run n times.
The function utilizing the select query uses O(n^2) because it performs O(n) in n times. The accumulator of tourDist by adding currentDistBtwn each time is down in O(n) now because the constant execution is run n number of times.

- O(n^2).

Therefore, overall the **calcTotalDist()** has a time of

- O(n^2).

5.
```
void studentpage::on_addSouvenir_button_clicked()
{
    Souvenir souv;                                                  1
    //name and campus
    QString name, campus;                                           1
    name = ui->souv_comboBox->currentText();                        1
    souv.souvName = name;                                           1
    campus = ui->selectCampus_comboBox->currentText();              1
    souv.campus = campus;                                           1


    //quantity and item cost
    int quantity = ui-> quantity_spinBox -> cleanText().toInt();    1
    souv.quantity = quantity;                                       1
    double itemCost = database.GetTotalCost(campus, name);          n
    itemCost = itemCost * quantity;                                 1
    souv.cost = itemCost;                                           1

    souvenirCart.push(souv);                                        1

    //update Cart table
    database.updateCartQuantity(campus, name, quantity);            n

    //displays cart in table
   if(sQry == "")                                                   1
   {
```

```
       sQry += "select souvCollege as 'College', souvTrad as 'Souvenirs',
souvCost as 'Cost', quantity as 'Quantity' "
                       "from Cart where souvCollege = '" +campus+ "' and
souvTrad = '" +name+ "'";                                                  1
   }
   else                                                                    1
   {
       sQry += " union select souvCollege as 'College', souvTrad as
'Souvenirs', souvCost as 'Cost', quantity as 'Quantity'"
                       "from Cart where souvCollege = '" +campus+ "' and
souvTrad = '" +name+ "'";                                                  1
   }

   showSouvCartTableView(database.loadSouvCart(sQry));                     n
   showTotalCost(itemCost);                                                1

}
1+1+1+1+1+1+1+n+1+1+n+1+1+1+1+1+1+n+1 = 3n + 16 = O(3n) = O(n)
```

All the initializations of Souvenir, int, double, and QString are constant time. In addition, name reads the string in from the souv_comboBox, campus reads the string in from the selectCampus_comboBox, and quantity reads the string in from the quantity_spinBox.

- O(1)

Variable itemCost is read in as a double by finding it from the database using the name and campus selected. Search uses “SELECT” statement in SQL, which traverses row by row in the database.

- O(n)

The Souvenir souv is pushed into SouvCart stack after initializing the values into souv in constant time.

- O(1)

updateCartQuantity uses the “UPDATE”…”SET” function from SQL which searches through the database for WHERE the campus and souvenir name match to set the quantity to the quantity chosen. Worse case if it has to go through all the rows in the database to find the desired item to quantify.

- O(n)

If this was the first souvenir added to the cart, the QString sQry would go use the select statement to construct the start of the query. Otherwise, it would be more than the first souvenir to add to the cart, and it would start with a “UNION” to combine the next select query. Both paths are constant time for concatenating the QString sQry.

- O(1)

showSouvCartTableView(database.loadSouvCart(sQry) will run a select statement that picks the desired rows from the query that outputs through the database of the Cart into the table view

- O(n)

showTotalCost displays the item cost into the totalCost label in constant time

- O(1)

Overall, the function **on_addSouvenir_button_clicked**() has a time of

- O(n)