# Assignment 2 Report – Chidiebere Umah

## Run Guide:

Use [make] to run build the classes

Use [make run] to run part 1

Use [make clean] to remove .class files

Use [make Graph] to display the graphs produced by the experiment

Use [make Experiment] to redo the whole experiment - Please note that this will take a few minutes to run the whole experiment.

# Design and implementation of OOP and data structures:

## Classes:

**Object Class: Generics**

Purpose : The purpose of this class is to store the term, the generic sentence and the confidence score

Attributes:

Term (String) : The variable storing a term in the knowledge base.

Sentence (String): The sentence describing the term

confidence_score (Double): The confidence score of the term

**BST implementation:**

- **Class: AVLTreeNode<>**

  Purpose : Node used to store the data of the AVLTree as well as the left and right child.

- **Class: AVLTree<>**

  Purpose : Class used to create an AVLTree data structure.

- **Class: GenericsKbAVLApp**

  Purpose : Manager Class the defines and run all the functions for Part1 as well as the methods used to get the experiment data

**Experiment Classes:**

- **Class: Experiment**

  Purpose : Class is used to run the whole experiment and create new data for the experiment each time it is run.

- **Class : Graph**

  Purpose : Class is used to create the graphs for the experiment using the data stored in the access database

## Interactions:

The **Generic** object class is used by the **GenericsKbAVLApp** manager class to store the term,sentence and confidence score.The **AVLTree** generic class interacts with the **AVLTreeNode** class that is used as nodes to create an AVLTree data structure. The **GenericsKbAVLApp** makes use of the AVLTree class. The **Experiment** class uses the **GenericsKbAVLApp** to create the data for the experiment and the **Graph** class to plot the graphs

# 2. Experimental Tests

### Trial test values and outputs (Part 1) :

The test values will be shown in the following format: **Test Term** - output

**Ephemeral** - ephemeral:An ephemeral is an insect (Confidence score: 1.00)

**Serendipity** - serendipity:A serendipity is good fortune (Confidence score: 1.00)

**Symphony** - symphony:Symphonies are musical compositions. (Confidence score: 1.00)

**Nostalgia** - nostalgia:Nostalgia is desire (Confidence score: 1.00)

**Labyrinth** - labyrinth:Labyrinths are mazes. (Confidence score: 1.00)

**Luminescence** - The term: Luminescence ,is not in the knowledge base

**Tenacious** - The term: Tenacious ,is not in the knowledge base

**Cascade** - cascade:Cascades are succession. (Confidence score: 1.00)

**Antiquated** - The term: Antiquated ,is not in the knowledge base

**Euphony** - The term: Euphony ,is not in the knowledge base

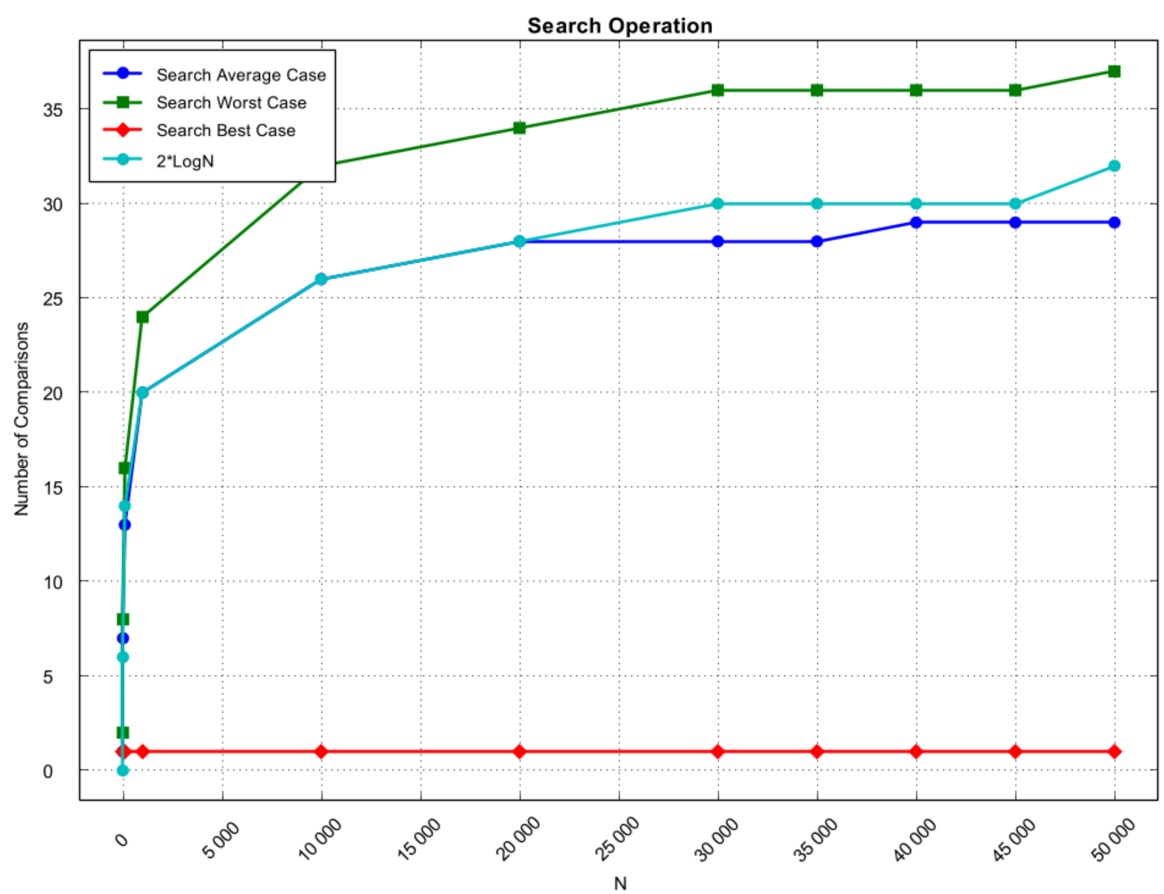
### Experiment Goal And Execution:

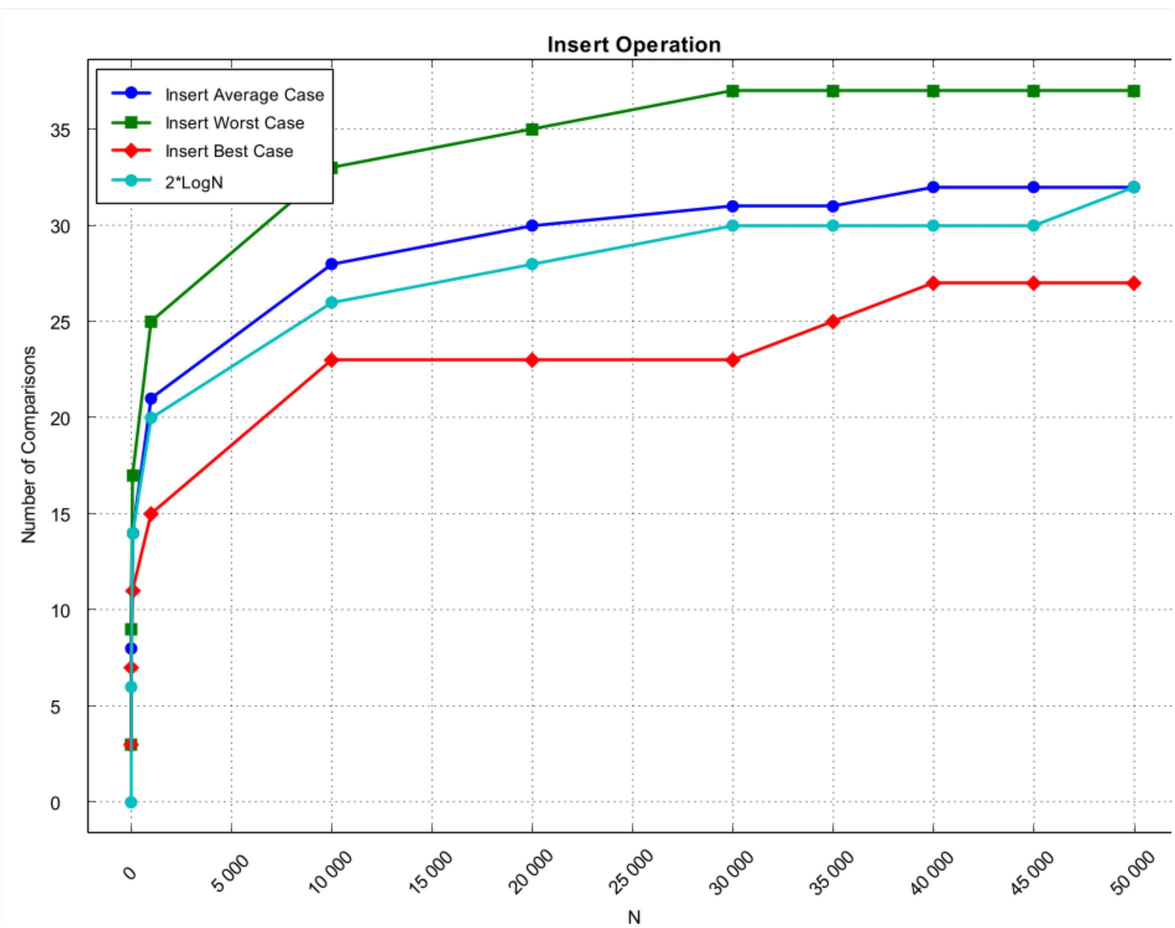
The goal of this experiment is to compare the theoretical time complexity of an AVLTree with the time taken in practice.

The execution of the experiment involved:

- Varying the number of elements inside the AVLTree by inserting a random subset of size N elements inside the AVLTree
- Counting the number of comparisons for both insert and search operations with all 5000 queries in the query file and loading these comparison inside an Access Database
- Getting the minimum,average and maximum number of comparisons for each size of N
- Plotting a graph using the minimum,average and maximum values and comparing these to a log base 2 graph

**Results:**


Search Operation
Search Average Case
Search Worst Case
Search Best Case
2*LogN
Number of Comparisons
N
0
5
10
15
20
25
30
35
0
5 000
10 000
15 000
20 000
25 000
30 000
35 000
40 000
45 000
50 000


Insert Operation
Insert Average Case
Insert Worst Case
Insert Best Case
2*LogN
Number of Comparisons
N
0
5
10
15
20
25
30
35
0
5 000
10 000
15 000
20 000
25 000
30 000
35 000
40 000
45 000
50 000

**Discussion of results:**

In the average and worst cases, searching an AVL tree mimics the logarithmic growth of (2*logN). This aligns with the theoretical time complexity of AVL tree search operation, as described by Big O notation. The best-case scenario when searching becomes even faster, achieving constant time O(1) if the search element happens to be the root node.

Similarly, for insertion, the average, worst, and even best cases all exhibit a time complexity close to 2*logN. This demonstrates that the practical performance of the AVL tree insertion function closely matches its theoretical efficiency.

# Description of creativity

In this project I made use of the following:

- The use of an Access Database to store experimental data
- The use of external library ucanaccess to read and write from the database in java
- The use of the java Collections class to shuffle AVL tree entries to randomize entry samples into the tree
- The use of external library XChart to plot graphs

# Summary statistics from git

0: commit b492cc1f23e73085b74ce7386435a4a5e2133c04

1: Author: Chidiebere umah <chidie.umah@gmail.com>

2: Date: Fri Mar 22 22:00:58 2024 +0200

3:

4: Added Part1 text file

5:

6: commit 28c3896161b6c8ed6ad5547ab2f98c698987b90c

7: Author: Chidiebere umah <chidie.umah@gmail.com>

8: Date: Fri Mar 22 19:52:57 2024 +0200

9:

...

73: Author: Chidiebere Umah <chidie.umah@gmail.com>

74: Date: Wed Mar 13 15:28:25 2024 +0200

```

Added Object class - Generics.java

commit 566fe01d2f389dbe59d3f82709b9ae05177f47db
Author: Chidiebere Umah <130582292+45Degreess@users.noreply.github.com>
Date: Tue Mar 12 11:31:27 2024 +0200

Initial commit
```