Objectives: The primary goal of this lab assignment was to implement and explore advanced operator overloading concepts in C++, including subscript, stream, and comparison operators, and apply them within a Point class. Additionally, the assignment aimed to implement and analyze the performance of the insertion sort algorithm for Point objects.

Learnings:

- **Subscript and Stream Operator Overloading:**
  - Implemented the **l2Norm** function within the Point class to calculate the Euclidean distance from the origin.
  - Successfully overloaded the subscript operator **[]** to work both as an l-value and an r-value within the Point class.
  - Implemented the stream out operator << for the Point class to display points and their Euclidean norms in a specified format.
- **Insertion Sort for Point Objects:**
  - Created arrays of Point objects of varying sizes and implemented the insertion sort algorithm using the < and > operators for sorting the Point objects.
  - Utilized the stream out operator to print the sorted list of points.
- **Performance Optimization:**
  - Measured the execution time for each value of **n** (array size) when performing the insertion sort on Point objects.
  - Constructed a graph showcasing the execution time (y-axis) versus the array size (x-axis) to analyze the scaling of the code as **n** increases.
  - Implemented an insertion sort for **int** types and compared its performance against the Point object-oriented code to understand any performance overhead incurred by object-oriented programming.

Challenges:

- Understanding the intricacies of overloading operators, especially the subscript and stream operators, required careful attention to handling both l-values and r-values.
- Optimizing the performance of the code, particularly for larger array sizes, posed challenges in balancing computational efficiency and readability in object-oriented code.

Key Notes:

- Operator overloading facilitates the creation of intuitive interfaces for classes by defining custom behavior for operators.
- Execution time analysis aids in understanding the scalability of code with increasing input sizes.

Conclusion: This lab assignment provided comprehensive insights into advanced operator overloading and performance analysis in C++. It offered practical exposure to optimizing code, utilizing custom operators for classes, and analyzing code scalability with varying input sizes.