VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI 590018

Project Report on

"ALGORITHM CALCULATOR"

By

PAVANKUMAR DEVARAJ NAIK(1BM24CS201)

PAVANKUMAR MUDGAL (1BM24CS202)

PRADEEP S HOSAMANI(1BM24CS205)

PRADEEP SHANKARGOUD BIRADAR(1BM24CS206)

Under the Guidance of

MONISHA H M

Assistant Professor, Department of CSE

BMS College of Engineering

Work carried out at

Department of Computer Science and Engineering

BMS College of Engineering

(Autonomous college under VTU)

P.O. Box No.: 1908, Bull Temple Road, Bangalore-560 019

2025-2026

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOPS with JAVA project titled "**ALGORITHM CALCULATOR"** has been carried out by PAVANKUMAR DEVARAJ NAIK(1BM24CS201), PAVANKUMAR MUDAGAL (1BM24CS202), PRADEEP S HOSAMANI (1BM24CS205), PRADEEP SHANKARAGOUD BIRADAR (1BM24CS206) during the academic year 2025- 2026.

Signature of the guide

MONISHA H M

Assistant Professor,

Department of Computer Science and Engineering

BMS College of Engineering, Bangalore

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, PAVANKUMAR DEVARAJ NAIK(1BM24CS201), PAVANKUMAR MUDAGAL (1BM24CS202), PRADEEP S HOSAMANI (1BM24CS205), PRADEEP SHANKARAGOUD BIRADAR (1BM24CS206), students of 3rd Semester B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this project work entitled "**ALGORITHM CALCULATOR**" has been carried out by us under the guidance of Monisha H M, Assistant Professor, Department of CSE, BMS College of Engineering, Bangalore during the academic semester Sep-Dec 2025. We also declare that to the best of our knowledge and belief, the project reported here is not from part of any other report by any other students.

Signature of the Candidates:

[illegible]

Pradeep.S.H

[illegible]

Table of contents

1.PROBLEM STATEMENT

Sorting is one of the most fundamental and widely used operations in computer science, as it plays a crucial role in organizing data efficiently for faster searching, processing, and analysis. Numerous sorting algorithms have been developed over time, and each algorithm differs in terms of its approach, time complexity, space complexity, and performance under different conditions. Choosing the appropriate sorting technique is essential for improving the efficiency of software applications that handle large volumes of data.

The problem addressed in this project is the lack of a simple and interactive platform to understand, implement, and compare different sorting algorithms using a practical programming approach. Hence, there is a need to design and develop a Java-based application called Algorithm Calculator that demonstrates the working of various sorting algorithms in a clear and structured manner.

The application should allow users to input a set of numerical data, select a desired sorting algorithm, and view the sorted output. Additionally, the project aims to help users understand the behavior and efficiency of different sorting techniques by comparing their performance. This project serves as an educational tool to strengthen the understanding of sorting algorithms while effectively applying Object-Oriented Programming concepts using Java.

2.INTRODUCTION

Sorting algorithms play a crucial role in data processing, data management, and optimization in computer science. They are extensively used in various fields such as databases, operating systems, search engines, and real-time applications where data needs to be organized efficiently for faster access and better performance. Properly sorted data improves the efficiency of searching algorithms and enhances overall system performance.

Different sorting algorithms have been developed to address varying problem requirements, and each algorithm differs in terms of its working mechanism, time complexity, space complexity, and suitability for different data sizes. Some algorithms perform well on small or nearly sorted datasets, while others are more efficient for large volumes of data. Therefore, understanding the characteristics and limitations of each sorting technique is essential for selecting the most appropriate algorithm in real-world applications.

The Algorithm Calculator project is developed to provide a clear, structured, and interactive platform for understanding and implementing various sorting algorithms using Java. The project allows users to input data, select a desired sorting technique, and observe the sorted output. By implementing these algorithms using Object-Oriented Programming concepts, the project not only strengthens algorithmic knowledge but also enhances practical Java programming skills. This project serves as an educational tool for students to bridge the gap between theoretical concepts and practical implementation of sorting algorithms.

3.OVERVIEW OF THE PROJECT

The Algorithm Calculator is a Java-based application designed to demonstrate the working of various sorting algorithms in a simple and structured manner. The main objective of the project is to help users understand how different sorting techniques operate and how their performance varies based on input data. The application provides an interactive environment where users can experiment with multiple sorting algorithms and observe their behavior.

The project follows a modular approach to ensure clarity, maintainability, and effective use of Object-Oriented Programming principles. Each sorting algorithm is implemented as a separate module, allowing easy comparison and extension of the application. The system accepts input data from the user, processes the data based on the selected sorting algorithm, and displays the sorted output in an organized format.

The application consists of the following major modules:

1. User Input Module

This module is responsible for accepting input data from the user. The user can enter a set of integers that need to be sorted. Proper input validation is performed to ensure correctness and avoid runtime errors.

2. Algorithm Selection Module

In this module, the user selects the desired sorting algorithm such as Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, or Quick Sort. Based on the selection, the corresponding sorting logic is invoked.

3. Sorting Execution Module

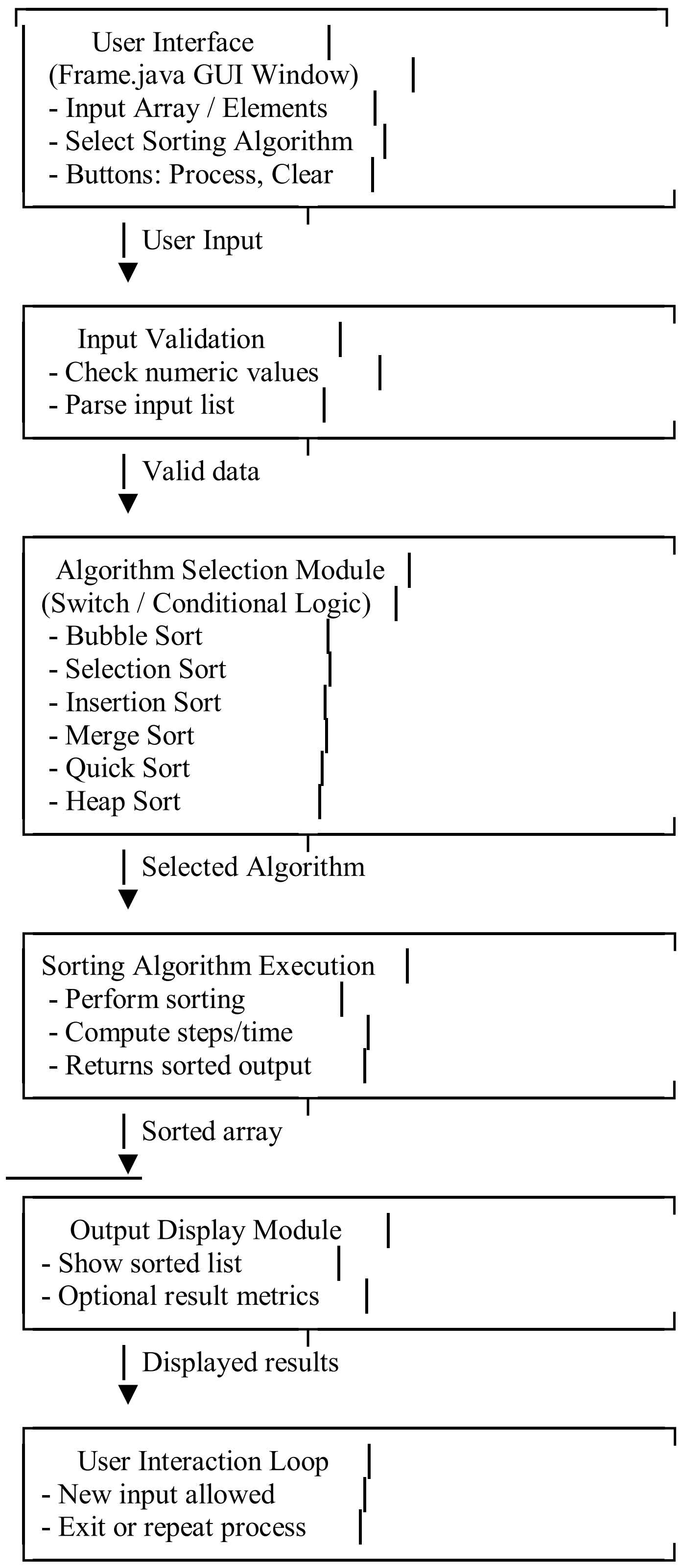
This module contains the core logic of all sorting algorithms. The selected algorithm processes the input data and sorts it in ascending order. Each algorithm is implemented using Java classes and methods, demonstrating Object-Oriented Programming concepts such as abstraction and polymorphism.

4. Output Display Module

The output module displays the sorted array to the user. It provides a clear representation of the final sorted data, helping users understand the result of the selected sorting algorithm.

The modular design of the project ensures ease of understanding, flexibility, and future enhancement, making it suitable for educational and academic purposes.

Block Diagram



4.Tools Used

The successful development and execution of the Algorithm Calculator project required the use of reliable programming tools and development environments. The tools selected for this project support Object-Oriented Programming, efficient coding practices, debugging, and execution of Java applications. Each tool plays a vital role in ensuring smooth development, testing, and implementation of sorting algorithms.

- **Programming Language: Java**
  Java is a platform-independent, object-oriented programming language widely used for developing robust and scalable applications. It provides strong support for data structures, exception handling, and modular programming, making it suitable for implementing sorting algorithms efficiently.
- **Integrated Development Environment (IDE): IntelliJ IDEA / Eclipse / NetBeans**
  These IDEs provide a user-friendly environment for writing, compiling, debugging, and executing Java programs. They offer features such as syntax highlighting, code completion, and error detection, which improve development productivity and code quality.
- **JDK Version: JDK 8 or Above**
  The Java Development Kit (JDK) includes essential tools such as the Java compiler and runtime environment. Using JDK 8 or above ensures compatibility with modern Java features and efficient program execution.
- **Operating System: Windows**
  The Windows operating system provides a stable platform for developing and running Java applications. It supports popular Java IDEs and offers a convenient environment for project development and testing.

5.OOPS CONCEPTS USED

The **Algorithm Calculator** project is developed using Object-Oriented Programming (OOP) concepts to ensure modularity, reusability, scalability, and ease of maintenance. The use of OOP principles helps in organizing the sorting algorithms efficiently and improves the overall structure of the application.

a) Class and Object

In this project, each sorting algorithm is implemented as a separate Java class such as `BubbleSort`, `SelectionSort`, `InsertionSort`, `MergeSort`, and `QuickSort`. These classes contain methods that define the logic of the respective sorting algorithms. Objects of these classes are created at runtime to access and execute the sorting methods. This approach allows better organization of code and demonstrates the practical use of classes and objects.

b) Inheritance

Inheritance is used to promote code reusability and establish a hierarchical relationship between classes. A base class or abstract class such as `SortAlgorithm` is created, which defines common properties or methods required for sorting. Specific sorting classes inherit from this base class and provide their own implementation of the sorting logic. This reduces code duplication and improves maintainability.

c) Polymorphism

Polymorphism is achieved through method overriding, where different sorting classes override a common method such as `sort()`. At runtime, the appropriate sorting algorithm is executed based on the user's selection. This enables the program to treat different sorting objects uniformly while allowing them to exhibit different behaviors, thereby enhancing flexibility and extensibility of the application.

d) Encapsulation

Encapsulation is implemented by declaring data members as private and providing controlled access through public methods. This ensures that the internal data of the classes is protected from unauthorized access and modification. Encapsulation improves data security and helps in maintaining a clear separation between data and logic within the application.

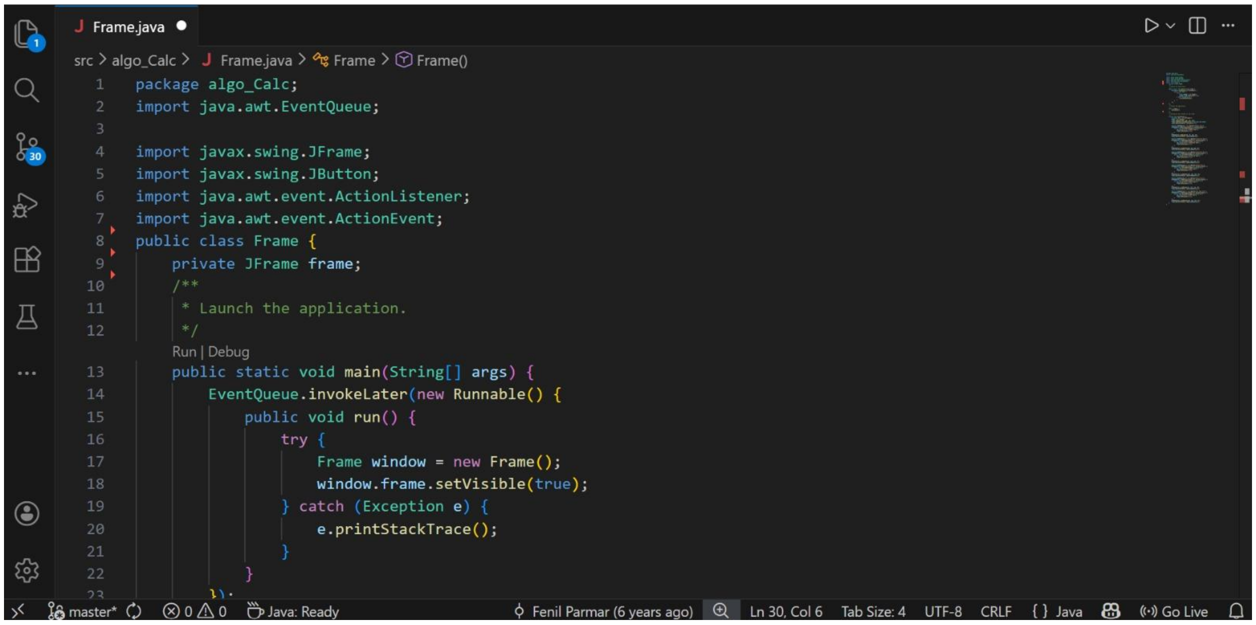
e) Abstraction

Abstraction is used to hide the internal implementation details of sorting algorithms from the user. Interfaces or abstract classes are used to define a common structure for all sorting techniques. The user interacts with the application without needing to understand the internal workings of each algorithm, which simplifies usage and enhances code readability.

f) Exception Handling

Exception handling is implemented using `try-catch` blocks to handle invalid user inputs and runtime errors such as incorrect data entry or array bounds issues. This ensures that the program does not terminate abruptly and provides meaningful error messages to the user. Proper exception handling improves the robustness and reliability of the application.

6.IMPLEMENTATION/CODE

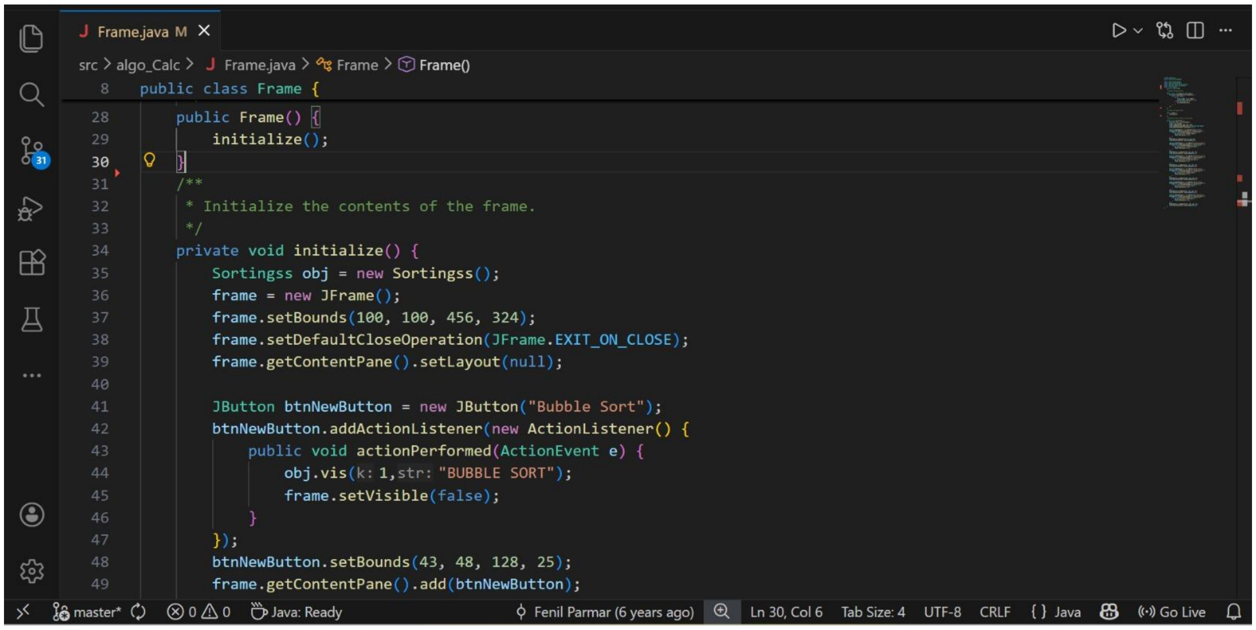


Frame.java

src > algo_Calc > Frame.java > Frame > Frame()

```java
package algo_Calc;
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
public class Frame {
    private JFrame frame;
    /**
     * Launch the application.
     */
    Run | Debug
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Frame window = new Frame();
                    window.frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
```

master* 0 0 Java: Ready Fenil Parmar (6 years ago) Ln 30, Col 6 Tab Size: 4 UTF-8 CRLF { } Java Go Live

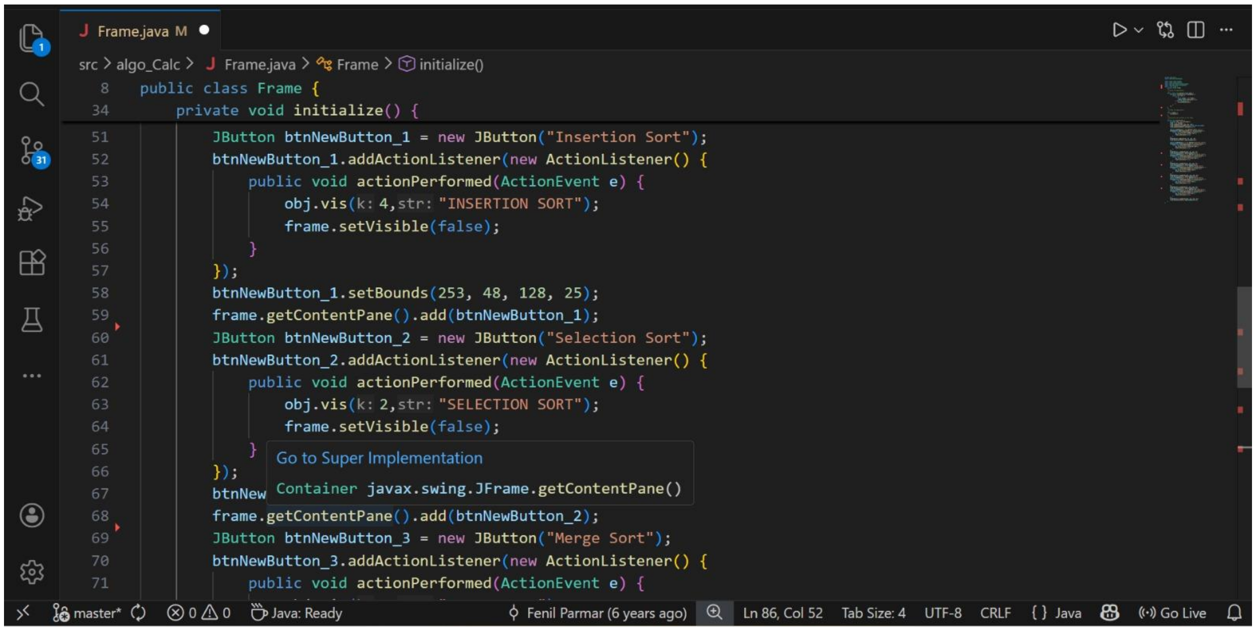


Frame.java M

src > algo_Calc > Frame.java > Frame > Frame()

```java
public class Frame {

    public Frame() {
        initialize();
    }
    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        Sortingss obj = new Sortingss();
        frame = new JFrame();
        frame.setBounds(100, 100, 456, 324);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(null);

        JButton btnNewButton = new JButton("Bubble Sort");
        btnNewButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                obj.vis(k: 1,str: "BUBBLE SORT");
                frame.setVisible(false);
            }
        });
        btnNewButton.setBounds(43, 48, 128, 25);
        frame.getContentPane().add(btnNewButton);
```

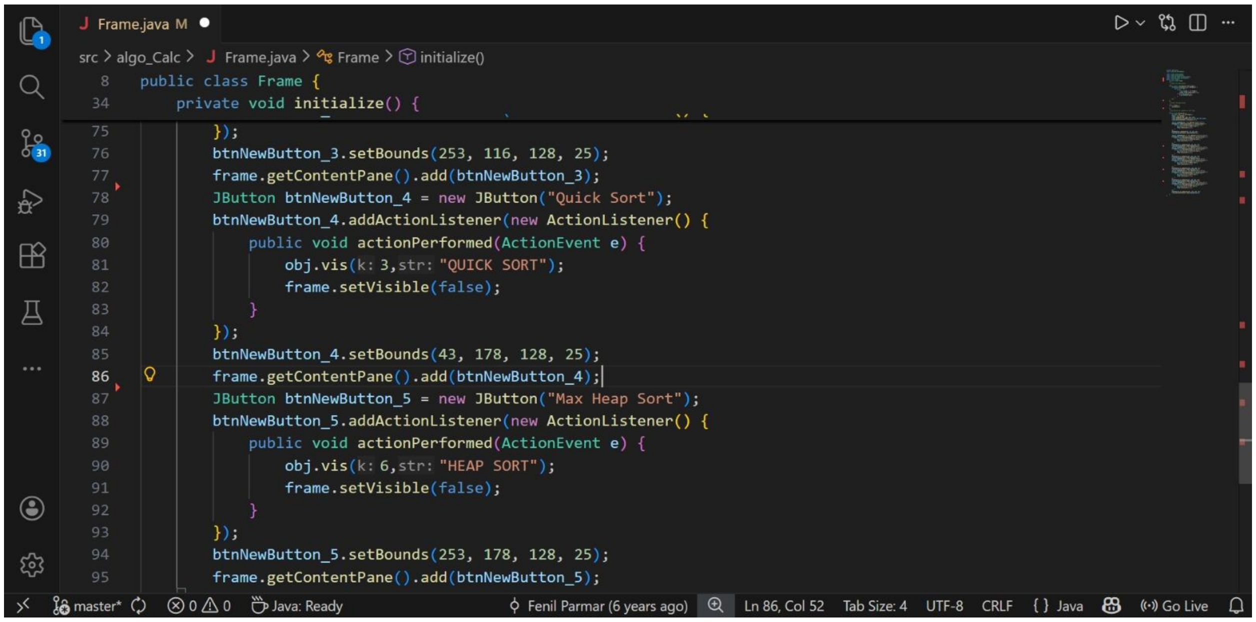
master* 0 0 Java: Ready Fenil Parmar (6 years ago) Ln 30, Col 6 Tab Size: 4 UTF-8 CRLF { } Java Go Live


Frame.java M
src > algo_Calc > Frame.java > Frame > initialize()
master* 0 0 Java: Ready Fenil Parmar (6 years ago) Ln 86, Col 52 Tab Size: 4 UTF-8 CRLF { } Java Go Live

```java
public class Frame {
    private void initialize() {
        JButton btnNewButton_1 = new JButton("Insertion Sort");
        btnNewButton_1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                obj.vis(k: 4,str: "INSERTION SORT");
                frame.setVisible(false);
            }
        });
        btnNewButton_1.setBounds(253, 48, 128, 25);
        frame.getContentPane().add(btnNewButton_1);
        JButton btnNewButton_2 = new JButton("Selection Sort");
        btnNewButton_2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                obj.vis(k: 2,str: "SELECTION SORT");
                frame.setVisible(false);
            }
        });
        btnNew
        frame.getContentPane().add(btnNewButton_2);
        JButton btnNewButton_3 = new JButton("Merge Sort");
        btnNewButton_3.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

Go to Super Implementation

Container javax.swing.JFrame.getContentPane()


Frame.java M
src > algo_Calc > Frame.java > Frame > initialize()
master* 0 0 Java: Ready Fenil Parmar (6 years ago) Ln 86, Col 52 Tab Size: 4 UTF-8 CRLF { } Java Go Live

```java
public class Frame {
    private void initialize() {
        });
        btnNewButton_3.setBounds(253, 116, 128, 25);
        frame.getContentPane().add(btnNewButton_3);
        JButton btnNewButton_4 = new JButton("Quick Sort");
        btnNewButton_4.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                obj.vis(k: 3,str: "QUICK SORT");
                frame.setVisible(false);
            }
        });
        btnNewButton_4.setBounds(43, 178, 128, 25);
        frame.getContentPane().add(btnNewButton_4);
        JButton btnNewButton_5 = new JButton("Max Heap Sort");
        btnNewButton_5.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                obj.vis(k: 6,str: "HEAP SORT");
                frame.setVisible(false);
            }
        });
        btnNewButton_5.setBounds(253, 178, 128, 25);
        frame.getContentPane().add(btnNewButton_5);
```

HEAP SORT

```
package algo_Calc;
class heaps {

    heaps() {
    }

    void heapify(int[] arr, int n, int i) {
        int largest = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;

        if (left < n && arr[left] > arr[largest]) {
            largest = left;
        }
        if (right < n && arr[right] > arr[largest]) {
            largest = right;
        }
        if (largest != i) {
            int temp = arr[i];
            arr[i] = arr[largest];
            arr[largest] = temp;
            heapify(arr, n, largest);
        }
    }
    void heapSort(int[] arr, int n) {
        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(arr, n, i);
        }

        for (int i = n - 1; i >= 0; i--) {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            heapify(arr, i, 0);
        }
    }
}
```

QUICK SRORT

```
package algo_Calc;

class quiks {
  quiks() {
  }

  void quickSort(int[] var1, int var2, int var3) {
    int var4 = this.partition(var1, var2, var3);
    if (var4 - 1 > var2) {
      this.quickSort(var1, var2, var4 - 1);
    }

    if (var4 + 1 < var3) {
      this.quickSort(var1, var4 + 1, var3);
    }
  }
  int partition(int[] var1, int var2, int var3) {
    int var4 = var1[var3];

    int var5;
    for(var5 = var2; var5 < var3; ++var5) {
      if (var1[var5] < var4) {
        int var6 = var1[var2];
        var1[var2] = var1[var5];
        var1[var5] = var6;
        ++var2;
      }
    }
```

```
        var5 = var1[var2];
        var1[var2] = var4;
        var1[var3] = var5;
        return var2;
    }
}
```

MERGE SORT

```
package algo_Calc;
class merg {
    merg() {
    }

    void merge(int[] var1, int var2, int var3, int var4) {
        int var8 = var3 - var2 + 1;
        int var9 = var4 - var3;
        int[] var10 = new int[var8];
        int[] var11 = new int[var9];

        int var5;
        for(var5 = 0; var5 < var8; ++var5) {
            var10[var5] = var1[var2 + var5];
        }

        int var6;
        for(var6 = 0; var6 < var9; ++var6) {
            var11[var6] = var1[var3 + 1 + var6];
        }

        var5 = 0;
        var6 = 0;
        int var7;
        for(var7 = var2; var5 < var8 && var6 < var9; ++var7) {
          if (var10[var5] <= var11[var6]) {
            var1[var7] = var10[var5];
            ++var5;
          } else {
            var1[var7] = var11[var6];
            ++var6;
          }
        }

        while(var5 < var8) {
          var1[var7] = var10[var5];
          ++var5;
          ++var7;
        }
        while(var6 < var9) {
          var1[var7] = var11[var6];
          ++var6;
          ++var7;
        }
      }
      void mergeSort(int[] var1, int var2, int var3) {
        if (var2 < var3) {
          int var4 = var2 + (var3 - var2) / 2;
          this.mergeSort(var1, var2, var4);
          this.mergeSort(var1, var4 + 1, var3);
          this.merge(var1, var2, var4, var3);
        }
      }
    }
```

BUBLE SORT

```
class BubbleSort
{
        void bubbleSort(int arr[])
        {
                int n = arr.length;
                for (int i = 0; i < n-1; i++)
                        for (int j = 0; j < n-i-1; j++)
                                if (arr[j] > arr[j+1])
                                {
                                        int temp = arr[j];
                                        arr[j] = arr[j+1];
                                        arr[j+1] = temp;
                }
        }
        void printArray(int arr[])
        {
                int n = arr.length;
                for (int i=0; i<n; ++i)
                        System.out.print(arr[i] + " ");
                System.out.println();
        }
        public static void main(String args[])
        {
                BubbleSort ob = new BubbleSort();
                int arr[] = {64, 34, 25, 12, 22, 11, 90};
                ob.bubbleSort(arr);
                System.out.println("Sorted array");
                ob.printArray(arr);
        }
}
```

INSERTION SORT

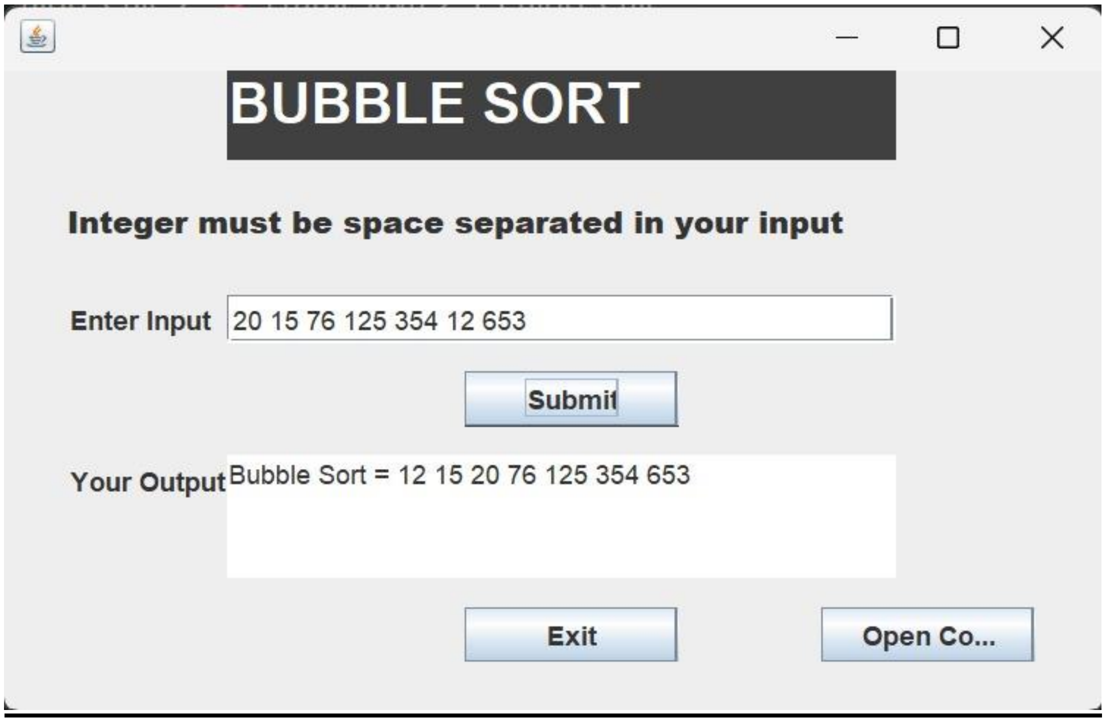
```
class InsertionSort {
        void sort(int arr[])
        {
                int n = arr.length;
                for (int i = 1; i < n; ++i) {
                        int key = arr[i];
                        int j = i - 1;

                        while (j >= 0 && arr[j] > key) {
                                arr[j + 1] = arr[j];
                                j = j - 1;
                        }
                        arr[j + 1] = key;
                }
        }

        static void printArray(int arr[])
        {
                int n = arr.length;
                for (int i = 0; i < n; ++i)
                        System.out.print(arr[i] + " ");

                System.out.println();
        }
        public static void main(String args[])
        {
                int arr[] = { 12, 11, 13, 5, 6 };

                InsertionSort ob = new InsertionSort();
                ob.sort(arr);

                printArray(arr);
        }
}
```

SELECTION SORT

```
class SelectionSort
{
        void sort(int arr[])
        {
                int n = arr.length;

                for (int i = 0; i < n-1; i++)
                {
                        int min_idx = i;
                        for (int j = i+1; j < n; j++)
                                if (arr[j] < arr[min_idx])
                                        min_idx = j;

                        int temp = arr[min_idx];
                        arr[min_idx] = arr[i];
                        arr[i] = temp;
                }
```

```
	}
	void printArray(int arr[])
	{
		int n = arr.length;
		for (int i=0; i<n; ++i)
			System.out.print(arr[i]+" ");
		System.out.println();
	}

	public static void main(String args[])
	{
		SelectionSort ob = new SelectionSort();
		int arr[] = {64,25,12,22,11};
		ob.sort(arr);
		System.out.println("Sorted array");
		ob.printArray(arr);
	}
}
```

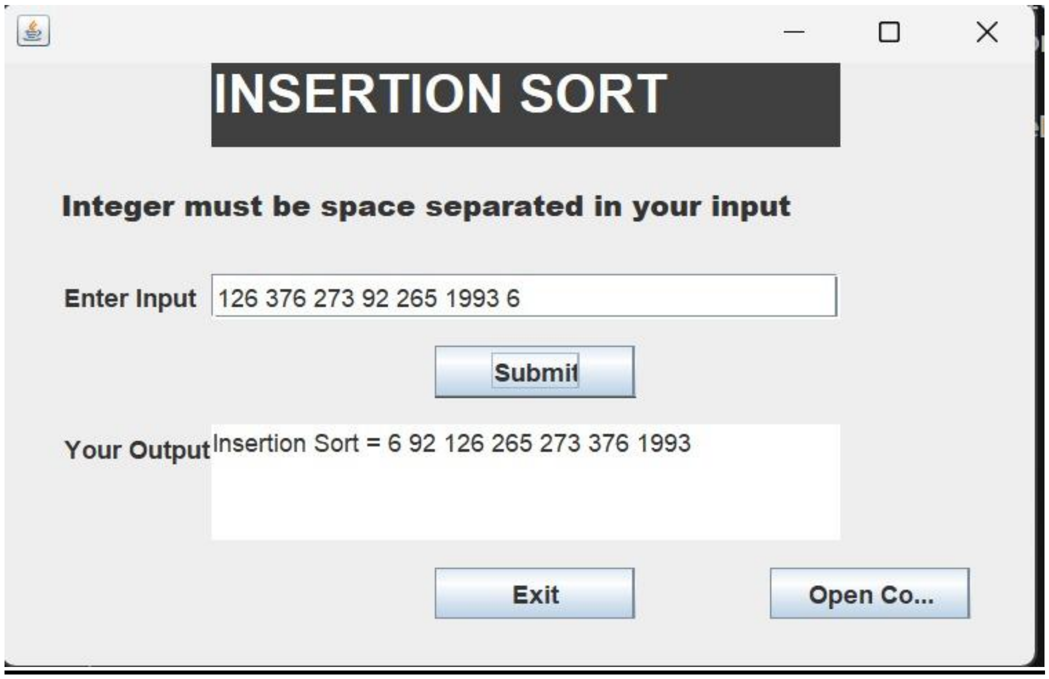
7.RESULTS/SNAPSHOTS



Integer must be space separated in your input

Enter Input 20 15 76 125 354 12 653

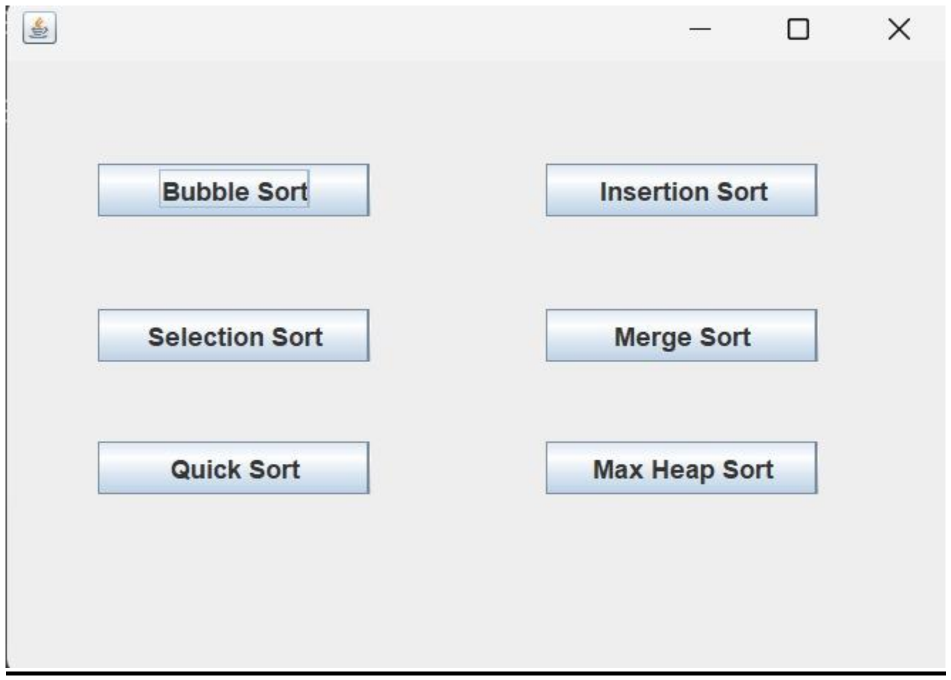
Your Output Bubble Sort = 12 15 20 76 125 354 653



Integer must be space separated in your input

Enter Input 126 376 273 92 265 1993 6

Your Output Insertion Sort = 6 92 126 265 273 376 1993


Bubble Sort
Insertion Sort
Selection Sort
Merge Sort
Quick Sort
Max Heap Sort

8.NEW LEARNINGS

The development of the Algorithm Calculator project provided valuable learning experiences and helped in gaining both theoretical and practical knowledge of Java programming and sorting algorithms. Working on this project enhanced understanding of various concepts that are essential for software development.

Through this project, a practical understanding of sorting algorithms such as Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort was achieved. Implementing these algorithms in Java helped in understanding their step-by-step working, efficiency, and differences in performance for various input sizes.

The project also enabled the effective application of Object-Oriented Programming **concepts** such as classes, objects, inheritance, polymorphism, encapsulation, and abstraction in a real-time application. This strengthened the ability to design modular and reusable code structures.

Additionally, the project helped in improving Java programming skills, including logical thinking, method design, and proper use of control structures. Writing and testing multiple sorting algorithms improved coding accuracy and confidence.

The implementation process provided hands-on experience in error handling and debugging techniques. Handling runtime errors, invalid inputs, and logical mistakes improved problem-solving skills and enhanced understanding of Java exception handling mechanisms.

Overall, this project contributed to developing analytical thinking, coding discipline, and a deeper understanding of algorithm design and software development practices.

9.FUTURE ENHANCEMENTS

The Algorithm Calculator project can be further enhanced to improve its functionality, usability, and educational value. Several additional features can be incorporated in the future to make the application more robust and interactive.

One of the major enhancements is the addition of searching algorithms such as Linear Search and Binary Search. Integrating searching techniques along with sorting algorithms will help users understand the complete data processing cycle and provide a more comprehensive learning platform.

The project can also be extended by developing a Graphical User Interface (GUI) using Java Swing or JavaFX. A GUI-based application would make the system more user-friendly and interactive by allowing users to input data, select algorithms, and view results through visual components such as buttons, text fields, and menus.

Another important enhancement is performance visualization using charts and graphs. Displaying execution time and comparisons of different sorting algorithms through bar charts or line graphs will help users visually analyze the efficiency of each algorithm for different input sizes.

Additionally, the application can be modified to support large datasets by optimizing memory usage and improving algorithm efficiency. This enhancement will enable the system to handle real-world data sizes and demonstrate the performance differences of sorting algorithms more effectively.

These future enhancements will increase the scalability, usability, and educational impact of the project and make it suitable for advanced learning and practical applications.

10.REFERENCES

- Herbert Schildt, Java: The Complete Reference
- www.geeksforgeeks.org
- www.javatpoint.com