



Gemma Qin

# Insertion and Search Algorithm Measurer with Raspberry Pi 4

## **Project Report**

Second-year Hardware Project

School of ICT

Metropolia University of Applied Sciences

30 February 2024

## **Abstract**

The project delves into the realm of embedded systems, focusing on the implementation and evaluation of data structures and search algorithms using the Raspberry Pi 4 Model B. The aim is to assess the performance characteristics of these elements and provide valuable insights.

Throughout the project, various data structures and search algorithms were implemented, performance measurements and comparisons were conducted. Challenges were encountered, particularly in debugging and interface integration, but were effectively managed through iterative refinement.

While the measurement system has limitations, it serves as a valuable tool for understanding data structures and the performance of search algorithms. Future directions include expanding to the addition of sorting algorithms and comparative analysis of processor performance across different platforms.

In conclusion, the project contributes to the advancement of embedded systems development by providing insights and tools for optimizing data structures and algorithms in resource-constrained environments.

# Contents

<b>1 Introduction.....</b>	<b>1</b>
<b>2 Theoretical Background.....</b>	<b>2</b>
<b>2.1 Data Structures.....</b>	<b>2</b>
2.2 Insertion and Search Algorithms.....	3
2.2.1 Sequential Search.....	3
2.2.2 Interval Search.....	4
2.3 Time Measurement.....	4
<b>3 Methods and Material.....</b>	<b>5</b>
3.1 Raspberry Pi 4 Model B.....	5
3.2 Frontend Development.....	6
3.3 Backend Development.....	7
3.4 Web Server.....	7
<b>4 Implementation.....</b>	<b>8</b>
4.1 Development in Windows Environment.....	8
4.2 Migration to Linux Environment.....	8
4.3 Migration to Raspberry Pi 4.....	9
4.4 Program Flow.....	9
<b>5 Installation.....</b>	<b>12</b>
<b>6 Conclusions.....</b>	<b>13</b>

## **1 Introduction**

The topic of the project centers around the implementation and analysis of various data structures and search algorithms within the context of embedded systems, specifically utilizing the Raspberry Pi 4 Model B. The project aims to address the need for efficient data structure and search algorithm implementations tailored for embedded systems. Embedded devices often face resource constraints such as limited processing power and memory, yet they are increasingly tasked with handling complex computations and data management tasks. Therefore, it becomes crucial to optimize the performance of these systems, particularly in scenarios where real-time data processing is required.

The primary motivation behind this project is to develop a comprehensive understanding of the performance characteristics of different data structures and search algorithms when deployed. By conducting thorough measurements and analysis, the aim is to provide insights into which combinations of data structures and algorithms are most suitable for various applications. The goals include evaluating the performance of different data structures and search algorithms, comparing their strengths and weaknesses, integrating a user-friendly web interface for easy configuration, and presenting measurement data in a visually appealing format for easy interpretation.

By achieving these goals, the aim is to contribute valuable insights and tools for developers and engineers working on embedded systems, ultimately enhancing the efficiency and effectiveness of such systems in diverse applications.

## 2 Theoretical Background

### 2.1 Data Structures

In the project, each data structure is implemented in its own class to ensure modularity, encapsulation, and ease of maintenance. A variety of data structures is deployed, each tailored to address specific requirements and scenarios encountered in embedded systems. These data structures play a crucial role in facilitating efficient data management and retrieval, thus impacting the overall performance of the system.

The data structures utilized in the project include:

1. **Array:** A contiguous collection of elements stored in memory, providing constant-time access to individual elements. Arrays are well-suited for scenarios where elements are accessed sequentially or by index.
2. **Circular Array:** A variation of the traditional array where the last element is connected to the first element, forming a circular structure. Circular arrays offer advantages such as efficient rotation operations and improved cache locality.
3. **Singly Linked List:** A collection of nodes where each node contains a data element and a reference to the next node in the sequence. Singly linked lists excel in scenarios requiring insertion and deletion at the beginning of the list, which times are constant.
4. **Doubly Linked List:** Similar to singly linked lists, but each node contains references to both the next and previous nodes in the sequence. Doubly linked lists provide efficient traversal in both forward and backward directions, making them suitable for scenarios involving bidirectional iteration.
5. **Custom Binary Tree:** A hierarchical data structure composed of nodes, where each node has at most two children: a left child and a right child. Binary trees facilitate efficient search, insertion, and deletion operations, with logarithmic time complexity for these operations when balanced.

6. **STL Binary Tree:** A binary tree implementation provided by the C++ Standard Template Library (STL), offering a standardized and optimized solution for various tree-based operations.

Each of these data structures is crafted to optimize performance and memory usage, taking into account the unique characteristics and constraints of embedded systems. By implementing each data structure in its own class, it ensures clear separation of concerns and facilitates easy integration into the measurement and evaluation framework.

## 2.2 Insertion and Search Algorithms

In the realm of data structures and search algorithms, the project delves into the implementation and evaluation of various techniques. Two fundamental approaches for insertion and search operations are sequential search and interval search.

### 2.2.1 Sequential Search

Sequential search, also known as linear search, is a basic method for finding a target value within a list or array. It involves sequentially checking each element in the data structure until the desired element is found or the end of the structure is reached. While simple to implement, linear search exhibits a time complexity of  $O(n)$ , where  $n$  represents the number of elements in the structure. This means that the search time increases linearly with the size of the data structure. [1]

For the project, linear search is utilized as the basis for evaluating the performance of linear data structures such as arrays and linked lists. By measuring the insertion time and search efficiency of linear search within these structures, the aim is to get insights into real-time data processing scenarios.

### **2.2.2 Interval Search**

Interval search, specifically binary tree search, offers a more optimized approach for locating elements within a sorted data structure. Binary tree search operates by recursively dividing the search interval in half, narrowing down the range of possible locations for the target element with each iteration. This results in a time complexity of  $O(\log n)$ , significantly faster than linear search for large datasets. [2]

In the project, binary tree search serves as the cornerstone for evaluating the performance of logarithmic data structures like binary trees. By measuring the insertion time and search efficiency of binary tree search, the goal is to assess its effectiveness in embedded systems, particularly concerning memory utilization and search speed.

## **2.3 Time Measurement**

To accurately measure the performance of insertion and search operations, the C++ `<chrono>` library's `high_resolution_clock` functionality is selected, which provides high-resolution time measurement, allowing us to precisely capture the duration of operations with minimal overhead in nanoseconds. By utilizing `<chrono>`, the execution times of insertion and search algorithms can be quantified across different data structures and search methods, facilitating a comprehensive analysis of their performance characteristics in embedded environments.

### 3 Methods and Material

#### 3.1 Raspberry Pi 4 Model B

At the heart of the embedded project lies the Raspberry Pi 4 Model B, shown in Figure 1, is a versatile and powerful single-board computer (SBC) renowned for its compact form factor and robust performance capabilities. [3]



*Figure 1. Raspberry Pi 4 with casing [4]*

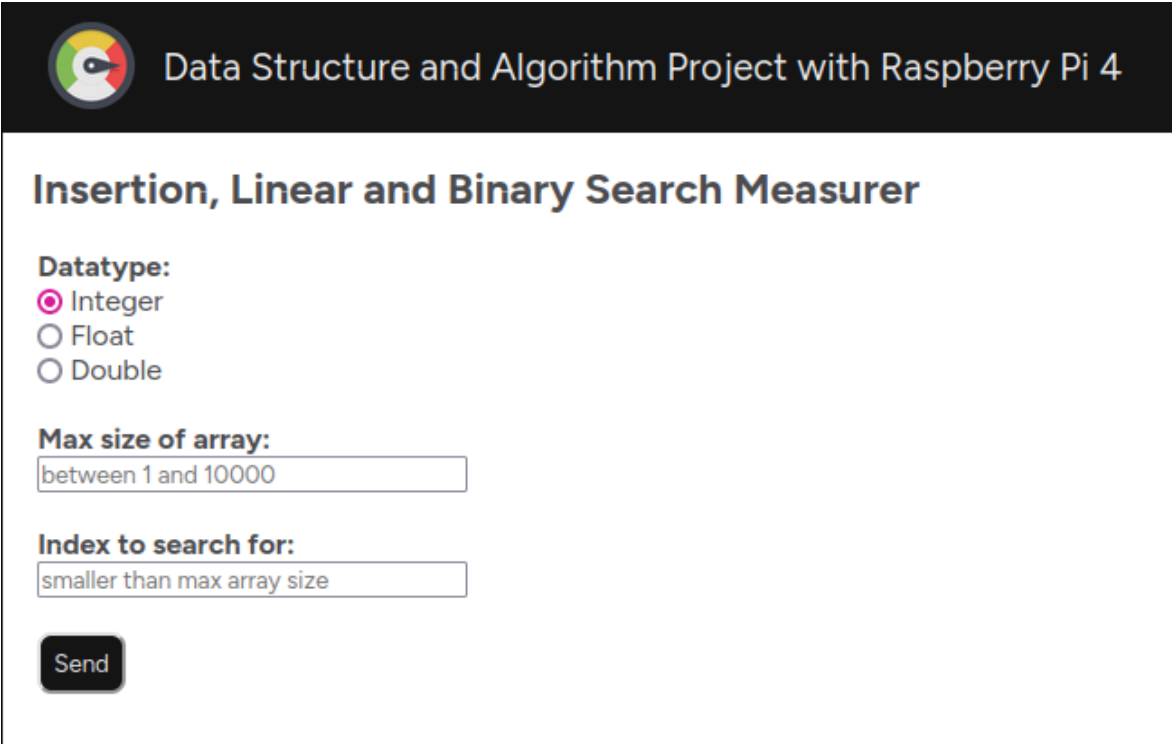
The Raspberry Pi 4 Model B serves as the primary computing platform for executing the data structure implementations, conducting performance



measurements, and interfacing with external systems. The Raspberry Pi 4 Model B provides the computational resources necessary for executing data structure implementations, conducting performance measurements, and interfacing with external systems.

### 3.2 Frontend Development

With the frontend development, the goal is to implement a user-friendly interface using HTML, CSS, and JavaScript. This interface, as displayed in Figure 2, enables users to input parameters such as data type, maximum array size, and the index of the element to be searched.



**Data Structure and Algorithm Project with Raspberry Pi 4**

### Insertion, Linear and Binary Search Measurer

**Datatype:**

☒ Integer  
☐ Float  
☐ Double

**Max size of array:**

**Index to search for:**

**Send**

*Figure 2. User Interface*

The simple yet intuitive layout ensures a streamlined user experience. JavaScript further enhances this experience by validating input values to ensure correctness before encapsulating parameters into JSON format and forwarding them to the Python script via an HTTP request.

### **3.3 Backend Development**

In the backend development phase, the focus is on the processing of the data received from the frontend interface and conducting measurements. To handle the input data effectively, the backend functionalities are implemented using a combination of Python and C++.

The Python script handles the reception of parameters from the frontend and the orchestration of the measurement process. The script calls the executable compiled C++ programme responsible for executing the measurement operations, passing the parameters received from the frontend as command-line argument. Additionally, the Python programme directs the output of the C++ programme, generated using `std::cout`, to a file for the transmission to the frontend.

### **3.4 Web Server**

For the web server setup, `lighttpd` (Lightweight HTTP Server) is selected along with Common Gateway Interface (CGI) for handling dynamic content generation. `Lighttpd` provides a lightweight and efficient web server solution, ideal for embedded systems with limited resources. [5] CGI enables the execution of external programs, allowing the integration of the Python and C++ backend functionalities with the web server. [6]

The configuration of `lighttpd` and CGI involves setting up the server environment, configuring CGI scripts, and establishing communication between the frontend interface and backend functionalities. Once configured, the web server serves as the intermediary between the user interface and the backend of the system, facilitating the transmission of parameters and measurement data in a secure and efficient manner.

## **4 Implementation**

In this section, the detailed implementation process of the project is discussed, starting from the development in a Windows environment to the final integration in a Raspberry Pi.

### **4.1 Development in Windows Environment**

Initially, the backend of the project was developed in a Windows environment using CLion IDE. Each templated class representing various data structures and search algorithms was developed in parallel with the main programme. Extensive testing was conducted for each class to ensure correctness and reliability before progressing to the next one.

Additionally, a templated class for random number generation was implemented to facilitate the generation of random numbers. To ensure uniqueness, the generated numbers are stored in a vector, and each new generated number is compared against existing elements to avoid duplication, which could potentially affect the outcome of measurements.

Following the development of the backend components, the frontend interface comprising HTML, CSS, and JavaScript was implemented using PyCharm IDE. The frontend was designed to provide a user-friendly interface for inputting parameters and a link to measurement result.

### **4.2 Migration to Linux Environment**

After the initial development phase, the development environment was migrated to a Linux environment, specifically Ubuntu. Here, the setup of the lighttpd web server and CGI was performed to facilitate communication between the frontend and backend components.

The integration of the backend and frontend components was orchestrated using a Python script. This script served as the intermediary between the frontend interface and the C++ backend programme. Upon receiving parameters from the frontend, the Python script passed them as arguments to the C++ backend programme, which then executed the measurement operations. The console output of the executable compiled programme was directed to a file by the Python script, which became accessible via link to the user on the frontend once the file creation occurred.

### **4.3 Migration to Raspberry Pi 4**

After confirming the successful execution of the project in the Ubuntu environment, the next crucial step was migrating the project to the Raspberry Pi 4 Model B, which operates on the Debian OS. The migration process involved replicating the development environment and setup on the Raspberry Pi 4 Model B, ensuring that all dependencies, configurations, and files were transferred accurately.

Once the setup was complete, the project was deployed on the Raspberry Pi 4 Model B, allowing for testing and validation to ensure smooth functionality in the embedded environment. By migrating the project to the Raspberry Pi 4 Model B and successfully deploying it, the project's adaptability and suitability for deployment in real-world embedded systems is demonstrated, reaffirming its effectiveness and reliability across different platforms.

### **4.4 Programme Flow**

The flow chart depicted in Figure 3 illustrates the programme's operational sequence, outlining the smooth interaction between the user interface, backend functionalities, and measurement processes.

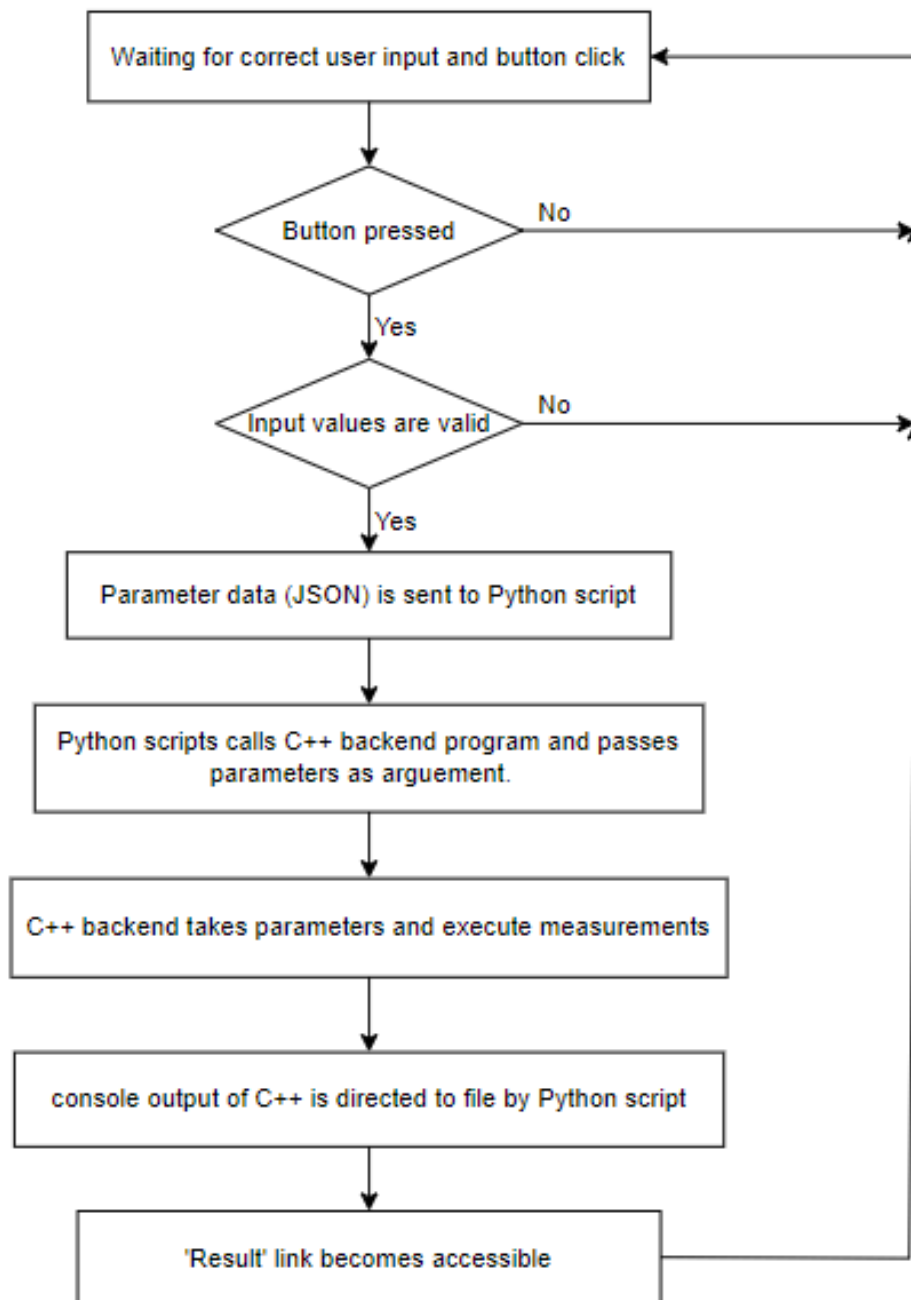


Figure 3. The flow chart of the programme

The user interface initiates the process by awaiting parameter inputs from the user alongside a button click to trigger the measurement operation. Upon button press, the JavaScript programme verifies the user inputs. If the inputs fall outside the specified bounds, the programme prompts the user to provide correct inputs and press the button again.

When valid inputs are provided, the parameter data is formatted into JSON format and transmitted to a Python script for further handling. Subsequently, the Python script calls the C++ backend programme, passing the parameters as arguments.

The C++ backend programme receives the parameters and executes the required measurement operations, such as insertion and search algorithms. The console output generated during this process is captured by the Python script and directed to a file for subsequent analysis and storage.

Upon completion of the measurement operations, the 'Result' link becomes accessible to the user via the user interface, allowing them to review the measurement outcomes.

The system in the meantime waits for new parameter inputs and another button press to initiate a new measurement process, thus ensuring a seamless and iterative flow for continuous interaction and measurement. This iterative flow design prioritizes user-friendliness and efficiency, facilitating smooth communication and execution of measurement tasks within the embedded system.

## 5 Installation

Setting up the Raspberry Pi OS and configuring the necessary dependencies is crucial for the successful deployment of the project. The step-by-step guide to ensure the proper installation and setup:

1. **Raspberry Pi OS Setup:** The latest version of the Raspberry Pi OS can be downloaded from the official Raspberry Pi website. Ensure that the OS is installed and configured.
2. **Dependency check:** Verify that the Debian-based system on the Raspberry Pi has the following dependencies installed:
  - make
  - cmake
  - gcc
  - lighttpd
3. **Project Setup:** Clone the [project repository](#) to the Raspberry Pi.
4. **Frontend Setup:**
  - Configure the lighttpd web server. Ensure that the lighttpd.conf settings and flags are correctly configured to allow access to the frontend files.
  - Copy the frontend files to the appropriate directory. In the original project's case this directory is located at /var/www/dsaproject/html.
5. **Backend Setup:**
  - Compile the backend.
  - Copy the compiled executable file (DataStructAlg) to the CGI directory.
6. **Verify** that the **lighttpd.conf settings** and flags are correctly configured to enable CGI execution and access to the backend executable. Ensure that the necessary permissions are set to allow execution of the CGI scripts.
7. **Restart** the **lighttpd** web server to apply the configuration changes.

With these installation steps completed, the Raspberry Pi should be set up and ready to execute the project. Ensure to test the functionality to confirm proper operation.

## 6 Conclusions

Throughout the project, we embarked on a thorough exploration of data structures and search algorithms tailored for small embedded systems, utilizing the Raspberry Pi 4 Model B as the computing platform. The objective was to assess the performance of these elements and contribute insights.

The project progressed smoothly, with successful implementation of various data structures and search algorithms. The performance of these data structures and search algorithms was evaluated, strengths and weaknesses were compared, and optimization strategies were explored. Though time constraints limited full exploration of all optimization avenues, the majority of primary goals were archived. Challenges, primarily in debugging and interface integration, were overcome through iterative refinement.

For future endeavors, expanding to include sorting algorithms and comparative analysis of processor performance across different platforms would enrich the project. Despite achievements, continuous improvement remains essential for advancing embedded systems development.



## References

- 1 GeekForGeeks. 2024. Linear Search Algorithm. Accessed: 26.02.2024  
<https://www.geeksforgeeks.org/linear-search/>
- 2 Wikipedia. 17 January 2024. Binary search algorithm. Accessed: 26.02.2024  
[https://en.wikipedia.org/wiki/Binary\\_search\\_algorithm](https://en.wikipedia.org/wiki/Binary_search_algorithm)
- 3 Raspberry Pi 4 Model B. Accessed: 26.02.2024  
<https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>
- 4 RaspberryPi.dk. Accessed: 26.02.2024  
<https://raspberrypi.dk/en/product/raspberry-pi-4-case-red-white/>
- 5 Wikipedia. 12 December 2023. lighttpd. Accessed: 26.02.2024  
<https://en.wikipedia.org/wiki/Lighttpd>
- 6 Wikipedia. 24 February 2024. Common Gateway Interface. Accessed: 26.02.2024  
[https://en.wikipedia.org/wiki/Common\\_Gateway\\_Interface](https://en.wikipedia.org/wiki/Common_Gateway_Interface)