Algorithms Workbench for Inf2B

Nevena Blagoeva

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh
2018

Abstract

AlgoBench is a dedicated learning aid tool designed to complement a course named Inf2B. Its objective is to assist the Inf2B students in learning runtime complexity analysis of algorithms and data structures taught throughout the course. This project is currently in its fourth development iteration and aims to extend the features of the tool by providing support for Priority Queue data structure, PDF generation, adding notes to the results and implementing flexible runtime queries. A major enhancement added to the Workbench in this development cycle is the package update for creating plots inside the tool. The evaluation results of the workbench strongly indicate that the tool will be greatly influential in helping the students develop better insight of runtime complexity analysis. In addition, further enhancements proposed by the author are discussed towards the end of the report.

Acknowledgements

My genuine and heartfelt gratitude to my supervisor, Dr Kyriakos Kalorkoti, for his attention, utmost dedication and suggestions throughout this project. Thank you for proposing this project and guiding me through it!

Many thanks to the previous developers Eziama E. Ubachukwu, Yufen Wang and Shalom Rachapudi for their great contributions to the software. Thank you for giving me the solid foundations and the opportunity to learn from your experience.

To my parents, who have always supported me and stand by me through my entire life, I say a big “Thank you”.

To the medical teams in Edinburgh and at home, led by Dr Daivid Millar and Dr Bojil Ilkov, thank you for taking care of me!

Finally, all glory to God, Father of my Lord and Saviour Jesus Christ, to whom I owe my strength, determination and very existence.

Table of Contents

Chapter 1

Introduction

Algorithms and data structures are an essential field in theoretical computer science. "They've been used for decades back to Alan Turing and the codebreakers, and beyond" [22]. The study of algorithms and data structures involves analyzing their computational complexity, scalability, efficiency etc [31]. These are of vital importance for fields like Operational Research, engineering, Mathematics and many more. Therefore, having solid understanding of their properties is necessary, especially in such a major center of Computer Science as the University of Edinburgh. The Inf2B course in the university, named "Algorithms, Data Structures and Learning" teaches some essential algorithms and data structures along with their runtime complexity. AlgoBench is a tool complementing the course by providing practical experience of the theoretical concepts of runtime analysis. The tool allows the students to execute different algorithms and data structure operations and observe their runtime charts for different input sizes.

This project is the continuation of the work carried out by Eziama E. Ubachukwu [47], Yufen Wang [49] and Shalom Rachapudi [41]. The aim of this development iteration is to enhance the algorithms of the Workbench, by providing support for the Priority Queue data structure [44]. Moreover, it further aims to add some non-trivial functionalities like PDF generation [41], adding notes to the results and flexible runtime queries. Another much needed enhancement, namely updating the chart package of the tool, was added to the application. The source code and the instructions for easy build with CMake [12, 41] can be found at https://github.com/nevenaBlagoeva/AlgoBench [37].

1.1 Outline of the report

The content of the report is organized as follows:

Chapter 2 presents the objective of the workbench, along with its architecture and main components. It lays out the aim of this development cycle and covers a literature review on related tools.

Chapter 3 starts by introducing the existing features and algorithms of the workbench which have been inherited from the previous three iterations. Then the main contributions of the author of this report are discussed in depth. It introduces the new package

used and the implementations carried out as part of this development cycle.

Chapter 4 presents the testing and evaluation of AlgoBench. It gives visual examples of the system in use and summarizes the User Evaluation results.

Finally Chapter 5 concludes the report and discusses some possible future enhancements of the workbench.

Chapter 2

Background and Literature Review

This chapter discusses the objective of the workbench and the motivation behind it. Furthermore, it gives a high-level overview of the architecture and the supporting mechanisms used for the data flow processes. It presents the aim of this development cycle and finally it briefly discusses tools similar to Algobench. The discussion in this chapter, if not otherwise specified, is based on the work carried out by Eziama Ubachukwu [47], Yufen Wang [49] and Shalom Rachapudi [41].

2.1 The objective of Algobench

Algobench was developed as a learning tool for students taking the Inf2B course. The system gives the students an opportunity to experiment with various algorithms taught in the course and develop their analytical thinking about their time and space complexity.

Algobench has great educational value because it gives students an opportunity of practical experiments, which will help them develop a better insight to the theory behind runtime analysis. Using the workbench students can observe how changes of the input settings to an algorithm affect significantly the runtime of the execution. This way they are able not only to validate what they have learned for a specific algorithm from the inf2B course, but develop analytical thinking of how the runtime changes under different circumstances. For example, by running linear search in AlgoBench students can choose to search an element which is always in the input array, a random one or an element which is never found in the input. By experimenting with similar settings for other algorithms the user can develop a better insight about worst, average and best-case runtime analysis.

2.2 Architecture and IPC Mechanism

A diagram of the architecture is presented in Figure 2.1. This figure is from the dissertation by Eziama Ubachukwu [47]. Since no changes to the high-level architecture have been applied, I present it in this document for the convenience of the reader. Pictures of the tool will be presented in Chapter 4.

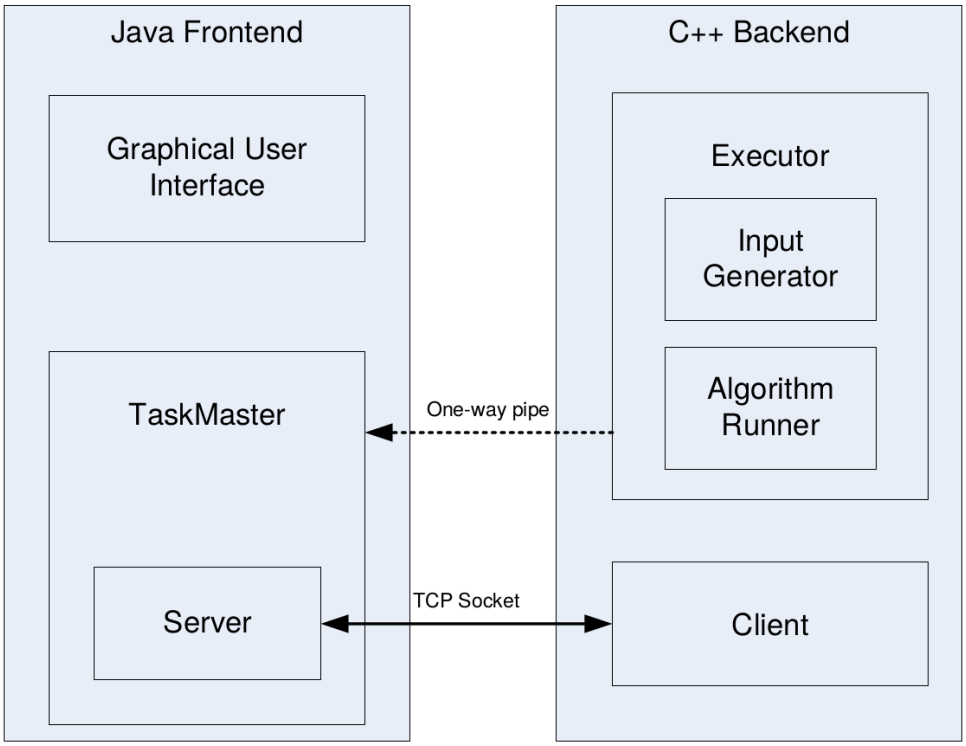


Figure 2.1: Architecture of AlgoBench [47]

It is important to note that by this development iteration the tool has grown to be a broad system, built upon various data structures, processes, interfaces and classes which come together in more than 30,000 lines. That is why in this section I will only discuss the high-level architecture.

AlgoBench consists of two separate parts – frontend and backend. The frontend is implemented in Java and the backend is written in C++. The rationale behind the choice of languages is discussed at the end of the section. Within the Java frontend, there are two main components – Graphical User Interface (GUI) and the `TaskMaster`. The first one is the interface which allows the user to interact with the system and the later is responsible for scheduling the task and starting the C++ Backend. Within the `TaskMaster` is the Server, which communicates with the Backend via a TCP (Transmission Control Protocol) socket. On the backend the main component is the Executor. It generates the input to the algorithm or uses an input file provided by the user. The input is then passed to the Algorithm Runner which runs the algorithm under the specific settings, previously set by the user in the frontend. The final result which is a collection of all the timings is send from the Client to the Server through the TCP socket.

In addition to the Client/Server communication, a second IPC (Inter-Process Communication) mechanism is employed – Anonymous pipes. While the algorithm executes and the timings are collected, live updates are being sent to the frontend through these one-way pipes. At the frontend the classes ProcessBuilder and Process are used to read these updates from the standard C++ output.

The rationale behind the choice of languages arises naturally. The C++ languages provides, more memory management and has pointers which refer to the memory [40,

47]. Also C++ is closely binded to the OS-system since it makes its calls directly to the OS functions. These factors make it faster than Java. Moreover, C++ frees memory alone, without the use of Garbage Collector, which sometimes slows down the program in order to free the memory. Java was chosen as a language to use at the frontend mainly because it provides good support for building the frontend. It has *Java Swing Framework* and an *Absract Windowing Toolkit*(AWT). Moreover, it is the preferred choice for large scale systems and implementing frontend features [15, 47].For these reasons AlgoBench frontend is written in Java and the backend in C++.

2.3 Aim of this development cycle

The aim of this development iteration is to enhance the application inherited from Eziama Ubachukwu [47], Yufen Wang [49] and Shalom Rachapudi [41]. This section covers the rationale behind the proposed enhancements, while Chapter 3 gives in depth analysis of the implementation.

The first enhancement provides support for Priority Queues [44]. Priority Queues are a data structure covered in the Inf2B course. Their implementation builds on several levels of abstraction [44], which are discussed in detail later on. Giving the students an opportunity to experiment with the most important operations – removing the maximum element and inserting a new element will greatly improve their intuition about the best, the worst and the average run-time, as explained later on. Moreover, the support for this data structure is full. By this we mean that apart from scheduling the task, the functionality of editing the Priority Queue task was developed. (The EditTask functionality was started by Shalom Rachapudi [41], however, due to time limitation this was provided for only a few of the algorithms AlgoBench supports, I extended it for Priority Queues).

The second enhancement was the PDF generation, which was originally started by Shalom Rachapudi [41]. The existing functionality did not cover all the algorithms and for those that it did the PDF generated was specializing to one model, inflexible for the different algorithms. This enhancement allows students to share the results of a task, use it for coursework analysis or tutorial exercise. The PDF generated contains the table of results related to the task and the graph produced after collecting all timings for different size inputs.

The third enhancement is adding notes to the result. The need for this functionality arises naturally, because after running a task it would be valuable if the students can share their observations with their tutor or classmates. This enhancement is achieved with the help of an external editor – LibreOffice [33]. It is closely related to the PDF generation and its development brings great value to the workbench with the full completion of PDF generation carried out.

The fourth enhancement is flexible runtime queries. Originally proposed by Shalom Rachapudi [41] this feature allows the user to execute code between two checkpoints. To improve user experience, the algorithm's pseudocode is displayed in the graphical user interface and the student is offered a choice of valid checkpoints (significant blocks of code). This functionality goes beyond the overall runtime of an algorithm, it allows the students to brake the runtime analysis of the algorithm into segments. It improves both their understanding of runtime analysis and of a specific algorithm.

The fifth enhancement is updating the `com.xeiam.chart` [3] package to `org.knowm.xchart` [2] package. The `com.xeiam.chart` is the package used for drawing the runtime graphs. It is no longer supported and was replaced by the more efficient and feature-rich `org.knowm.xchart` package. Even though the new package has been available for a few years, due to time constrains and the scale of this task, the workbench has not previously been updated with the new one.

The implementation, conceptual problems and more details about the enhancements are discussed in-depth in Chapter 3.

2.4 Related Work

This paragraph provides summary of the literature reviews done by Eziama Ubachukwu [47], Yufen Wang [49] and Shalom Rachapudi [41]. There are many tools designed to help students develop better understanding of algorithms and data structures. Most of them are animation tools which visualize the steps of executing an algorithm and the core operations of various data structures (linked lists, queues etc.). As identified by [47], *Vedya* is an educational tool developed at Universidad Complutense de Madrid, Spain. It is an interactive tool which uses animations to teach algorithms and data structures [42]. Another visualization tool is *Data Structure Visualization*, developed at University of California, it is a web service that provides visualization of well-known data structures, their associated operations, and also supports algorithm animations [16, 41]. There are many similar visualization tools such as Data Structures Learning (DSL) [5, 49], SPEED [20, 47], Visualization in Data Structure and Algorithms (VIDSAA) tool [11, 49], VisuAlgo [21, 41] etc. Similar visualization tools are common, however, tools which demonstrate runtime computational complexity as it is with *Algobench*, are a much rarer find. One such tool, which bares similarity to the workbench is *trend-prof* [19, 47]. It is a profiling tool which tries to analyze blocks of code in order to predict computational complexity. Unlike Algobench, this tool has more theoretical approach which uses the number of times a given block of code is executed, while Algobench provides real-world timing of given algorithms.

This paragraph summarizes the research carried out by me in this development cycle on tools of similar to Algobench character. As mentioned before there are many visualization tools, designed to assist learners in algorithms and data structures. One such tool is *Algorithm Visualizer* which is an impressive, new (2017), open source web service tool [46]. It supports various algorithmic groups - trees, searching, sorting, graphs etc. It provides the code of the algorithm inside a web editor, has a printing console and also does animations of the data structure used by an algorithm. The user has the opportunity to alter the code inside the editor and run it. During execution the currently active line of the code is highlighted and the console provides helpful prints, while the animation of the data structure gives a visual representation of the current swaps, comparisons, insertions or any other respective operations. Another visualization tool is *SORTING* [1]. As the name suggests it supports only sorting algorithms, similar to *Algorithm Visualizer* it provides animation of the relevant operation. *SORTING* allows the users to run several algorithms at once and presents separate bar charts of inversions during each operation. *Algorithm and Data Structure Visualisation (ADV)* is another visualization tool aiming to help students understand how algorithms

and data structures work. It has been developed by University of Edinburgh students and it is available from the Inf2B course web page [10]. Even though it provides analysis of some form, neither of the two systems shares the aim of the workbench, which is runtime complexity analysis in real time.

In the rest of the report technical literature and other useful sources are presented in the discussion.

Chapter 3

Existing features and new Implementation

3.1 Features and Functionalities of AlgoBench

This section contains features and functionalities developed by Eziama Ubachukwu [47], Yufen Wang [49] and Shalom Rachapudi [41]. This is the version of the tool inherited by the author of this report and also a starting point of this development cycle.

3.1.1 Existing Algorithms

Algorithm Group	Algorithm
Sort	Quick sort, Heap sort, Insertion sort, External Merge sort, Internal Merge sort
Graph	Breadth-First Search, Depth-First Search
Hash	Hashing
Search	Linear Search, Binary Search
Tree	Binary Search Tree

Table 3.1: Algorithms supported by AlgoBench. Table adapted from [41]

It is important to acknowledge the work carried out by the previous developers and say that as part of their masters thesis they have built the system to support various algorithms and some non-trivial functionalities. Table 3.1 contains a summary of the algorithms supported by the workbench. The main contributors to the algorithm support are Eziama Ubachukwu [47] (original developer) and Yufen Wang [49]. In the third development iteration Shalom Rachapudi [41] enhanced the tool by adding Binary Search Trees and other complex functionalities.

As mentioned in section 2.2 AlgoBench allows users to customize the input to the tool, this is done via three-stage `java.swing` input panels, specific to every algorithm. This way the students are able to configure settings like: input size, number of experiments per input size, input range, step size etc. The settings are used when generating

input at the backend. Some of the algorithms also support reading input from .csv or .txt file. In cases when the user decides to provide their own input, if it meets the workbench specifications, the system proceeds with exactly the same operations and provides the same functionality.

3.1.2 Existing Features

The workbench supports runtime analysis of algorithms through runtime charts, tab-separated values (TSV), real time messages and execution progress report [47, 49]. When executing an algorithm the output is stored in an $N \times M$ matrix where N is the number of execution steps and M is the number of runs of a task. The *runtime chart* is generated based on this data. The *tab-separated value* feature stores the $N \times M$ matrix in a tabular form and allows the user to export it in .tsv format. The *Execution progress report* named is a dynamic feature which consists of information such as percent completed tasks, memory footprint and current input size. *Real time messages* are concise messages informing the user of the current step performed at the backend [41]. Step, in this context, refers to generating input, building heaps, sorting an array, timing message or any other relevant information the developer has provided at the backend.

A significant feature added in the second development iteration [49] is comparing algorithms from the same Algorithmic group. Comparison is an important step in learning which allows students to observe clearly how changes of the input settings to an algorithm affect the output chart and table.

Another enhancement, added in the third development iteration [41], is editing an existing task. This functionality allows the user to modify a few parameters of an existing task, without going through the initial set up panels. This is achieved with the help of `JDialog` which allows the parameters to be reconfigured in a single step. At the end the user is offered to either replace the task they modify or to create a completely new one. This enhancement contributes to a better user experience and improves the interaction with the tool.

Building the software with CMake [12] is an enhancement Shalom Rachapudi [41] added to the workbench. CMake is a platform-independent building tool, which provides support for both C++ and Java. Even though, this functionality does not benefit the users directly, it is of great help to the developers who are building the software from source. With the help of `CMakeLists.txt` and `MANIFEST.mf` files, both the Backend C++ code and the Frontend GUI source code are handled, the jar file for the front end is generated, the Java GUI is linked with the libraries and at the end the compiled Backend and Frontend are integrated. This enhancement makes the tool independent of NetBeans, which was previously used to build and run the application.

PDF generation was started by Shalom Rachapudi [41] and developed further in this iteration. Following the five step method described in [6, 41] Shalom has created a `PDFGeneration` class that generated PDF from XSL-FO file. In his version of the system the PDF functionality only works for a few algorithms. The file contains the runtime chart and a table of features common to the group of algorithms, or sometimes a few groups. It populates the table with the elements they have in common: input range, step-size, scheduled tasks etc. The PDF enhancement is essential for students,

it allows them to gather all the information for an experiment in one place and later use it for a coursework assignment, tutorial or exam revision. For this reason it needed a lot of further work which is described in section 3.3 below.

Due to the scale of this project, bugs are inevitable. The version of the tool inherited at the beginning of this developments cycle also contained a bug related to the Hash task. It is further discussed in section 3.3.2.

3.2 New Package Used

3.2.1 XChart

This section features the discussion of one new tool, namely `org.knowm.xchart` [28], which was integrated into the workbench in this development cycle. The package is a light-weight library used for plotting data into charts and customizing their style. Some of the technical features of `org.knowm.xchart` are discussed below.

The knowm xchart package [28] is an open-source, feature-rich and light weight Java library for plotting data. It is focused on simplicity and is easy to use, designed to go from data to chart in the least amount of time. Only a few lines of code are needed to save or display a default chart [29]. The library gives the developer the option to style the chart and define most of its features whilst providing the freedom of choosing between many types of graphs.

Compared to the its predecessor, namely the `com.xeiam.xchart` package [50], the new `knowm` library provides much richer functionality [28]. Additionally to the Area, Line, Bar and Scatter charts it provides support for Histogram, Donut, Pie, Bubble charts, to name a few. As mentioned in [29], its extensive customization gives the developers freedom to adapt the chart to their objectives, whilst its high resolution chart export improves user experience. One of the new and most impressive features the package has to offer are the real-time charts. Unlike `xeiam`, `knowm` has active support, which combined with the small `.jar` file and the absence any additional dependencies make it a desirable choice [29]. Moreover, as discussed by Anshu Shukla, Shilpa Chaturvedi and Yogesh Simmhan in [43], it is one of the Java chart libraries that is very useful for certain web applications. This is relevant to the future plans for the workbench discussed in Chapter 5.

A simple example of building a basic chart is shown in Figure 3.1. The three necessary objects for building a chart with `knowm` are the Builder, Styler and the data Series [28]. In the figure logarithmic data points are generated as input. They are stored inside two `ArrayLists` and eventually at the end fed into the `XYChart` object. For building the chart is used the class `XYChartBuilder` and for customizing it the class `XYStyler` [29]. The `knowm` library is complex, however, very well organized. It provides five main chart types each of them uses different Builder, Styler and Series [30]. The detailed implementation of replacing the old, deprecated `com.xeiam.xchart` package [50] with the new `org.knowm.xchart` is discussed in depth in Section 3.3.5.

```
public XYChart getChart() {

    // generates Log data
    List<Integer> xData = new ArrayList<Integer>();
    List<Double> yData = new ArrayList<Double>();
    for (int i = -3; i <= 3; i++) {
      xData.add(i);
      yData.add(Math.pow(10, i));
    }

    // Create Chart
    XYChart chart = new XYChartBuilder().width(800).height(600).
                                         title("Powers of Ten").
                                         xAxisTitle("Power").
                                         yAxisTitle("Value").build();

    // Customize Chart
    chart.getStyler().setChartTitleVisible(true);
    chart.getStyler().setLegendPosition(LegendPosition.InsideNW);
    chart.getStyler().setYAxisLogarithmic(true);
    chart.getStyler().setXAxisLabelRotation(45);

    // Series
    chart.addSeries("10^x", xData, yData);

    return chart;
  }
```

Figure 3.1: A basic chart using org.knowm.xchart. Taken from [29]

3.3 Current Development Iteration

This section focuses on the features, functionality and enhancements added to the tool as part of this project. Unless otherwise explicitly specified all the work described below has been carried out in the current development iteration.

3.3.1 Data Structure: Priority Queues

This section discusses the addition of Priority Queues to the algorithms shown in table 3.1 of section 3.1.2. Priority Queues (PQ) [44] are an abstract data type which resembles the traditional Queue data structure, however, each element of the priority queue has an associated priority. When operating on a PQ, the elements with highest priority are processed before any other elements from the queue.

There are many ways one can implement Priority Queues - unsorted list of elements, which is searched sequentially when the element with the highest priority has to be found, AVL trees [9] or Heaps [32]. The implementation used in this report is Heaps, because the lecture notes that are used as part of the Inf2B course discuss the Heaps implementation of Priority Queues [25]. This is a good choice for a data structure, since the methods every Priority Queue needs to have - `insert_with_priority` and `remove_highest_priority_element` - can be implemented in an efficient way.

Inside the AlgoBench environment removing the element with highest priority is a well defined task, which does not depend on many additional parameters, apart from range of the input and the step size. However, for an insertion the students can pick between using AlgoBench-randomly generated data element, a custom element and a worst-case element. This implementation not only allows the users to build priority queues but also helps them develop a better understanding of how the runtime of insertion changes, depending on the priority. As with all of the other algorithms, while executing, the users are presented with their initial input settings and the execution progress report mentioned in section 3.1.2 – current input size, memory footprint, number of scheduled tasks etc.

An additional enhancement to the priority queues is the functionality of editing a PQ task. This feature is currently available for only a few algorithms, because it requires a separate implementation for each of them. It is now finalized for priority queues as well. With this last enhancement the priority queue support is complete and the students can use the new data structure to explore all of the additional functionalities provided by the workbench.

One major problem encountered while implementing priority queues inside the C++ backend is timing. Both `insert_with_priority` and `remove_highest_priority_element` perform at most $\log_2 n$ operations where n is the size of the queue. Collecting times in such a small interval on a standard desktop computer is tricky and requires a `nano timer`. On the one hand, theoretically this issue can be approached by increasing the size of the queue, however, this would require doubling the size of the heap in order to add just 1 more computational step which is way too little improvement. Creating queues of hundreds of billion nodes on a standard PC machine is unrealistic and leads to memory allocation errors [45]. For the reasons mentioned above timing the PQ operations needs to be perfect - no local variables assignments, conditional statements or prints inside the timed segments of code. The timing was implemented to occur inside of the `PriorityQueue` class and then returned as an output by the `operator()()` public method. It is important to note that the other algorithms and data structures make use of the `AlgoTimer` structure inside the `main.cpp` class, however, due to the refined timings in priority queue tasks, this functionality was not used and the different approach was adopted.

3.3.2 PDF generation

As discussed in Chapter 2 PDF generation was a task started by Shalom Rachapudi [41]. Even though there were some developments made – discussed in section 3.1.2, more attention to the task was needed in order to provide a complete and coherent system for the users.

To generate a PDF file the workbench uses the basic pattern laid out in Apache FOP [7]. The method is concise and can be logically divided into 5 steps. It starts with creating a new `org.apache.fop.apps.FopFactory instance` and an `OutputStream`, used respectively for configuring the PDF settings and writing the generated PDF document. From the `FopFactory` a `Fop` instance is created. The XML code and its relevant XSL file are set up for XSLT transformation, which results in `FO` events file. The events are then piped through the `Fop` instance and the XSLT transformation is started (Figure 3.2).

Figure 3.2: XML to PDF process [7, 41].

Once established, the method described above performs as expected, however, it uses XML and XSL files in order to configure the data and the design of the resulting PDF. Although I have used the same pipeline described above, there are some key

differences in building the XSLT file, the XML buffer and the data path for Binary Search Tree task. The system inherited at the beginning of this development iteration did not provide PDF generation for all existing algorithms from Table 3.1. And for those that it did, the XSL file which is used as an intermediate step between the XML code and the FOP instance which later will result in a PDF was limited to a fixed set of features, generalizing to a few algorithms at once. This is one possible implementation [41], however, the algorithms supported by the workbench are too diverse to fit one model. For instance, if arguments such as `Maximum recursion depth` and `Pivot position` make sense for `Quicksort` they do not represent a valid argument in `Linear` or `Binary search`. The appearance of empty fields which are irrelevant to the current task would be frustrating to the users. In order to achieve the desired outcome of a consistent system the XSL file was altered to insert a field in the PDF document only if an actual argument was provided when generating the `Task` object (used for executing the algorithm). Such conditional parsing of the XML fixes the issue with the irrelevant fields. References to the language and some new features used in the development process can be found in [27], a good guide to XSLT and XPath by M. Kay.

As mentioned earlier, the PDF functionality was not supported for all existing algorithms. The Hashing and Tree algorithmic groups were a particular issue. The problem with the hashing was a bug inherited from the previous development cycles. The C++ backend was not printing properly the stringstream containing the Buckets. This was due to a problem inside the constructor's member initializer list. The proper way of using member initializer lists is described in [13]. Once fixed, the dataflow from the backend was again available at the frontend and building the XML was a matter of configuring the XML string buffer. This bug fix is important when updating the chart's package and I will come back to it in section 3.3.5. The issue with the Tree algorithmic group was of more complex nature. When implementing Binary Search Threes, the previous developer had created an interface `ITreeSubPanel` which is later implemented by the `TreeTaskSubPanel` class, where `TreeTaskSubPanel` extends `javax.swing.JPanel` [41]. This individual to a task interface design is a very good software engineering approach. It brings clarity and structure to the code [35]. However, since the `TreeTaskSubPanel` class is initialized once, after running the task, relying only on the JPanel variables to hold the data is inconvenient and can lead to bugs in other functionalities. For instance, the tool has the functionality to save a task, a saved task is a `TaskMaster` object which contains all the information previously used to execute an algorithm, along with the results received from the C++ Backend. If the data received from the Backend is not present in the `TaskMaster`, when saving a result it is going to be lost. This will interfere with functionalities like generating a PDF from saved task. To fix this, I implemented the API needed by the `TaskMaster` class to access the Binary search tree data coming from the Backend.

Having overcome these two challenges, to complete the PDF functionality I changed the XML generation inside the `GeneratePDF` class – this allows the XML string buffers for different algorithms to receive their data from the task objects. The final step was enhancing the XSL file – which plays a significant part in creating the PDF. I provided the XSL structures for the new features which the `TaskMaster` could now obtain from the backend, along with the conditional inclusion of fields which takes only parameters initialized in the Task object [27]. This resulted in appropriate PDFs of all algorithms

AlgoBench supports. Figure 3.3 shows a pdf for the tree based task.

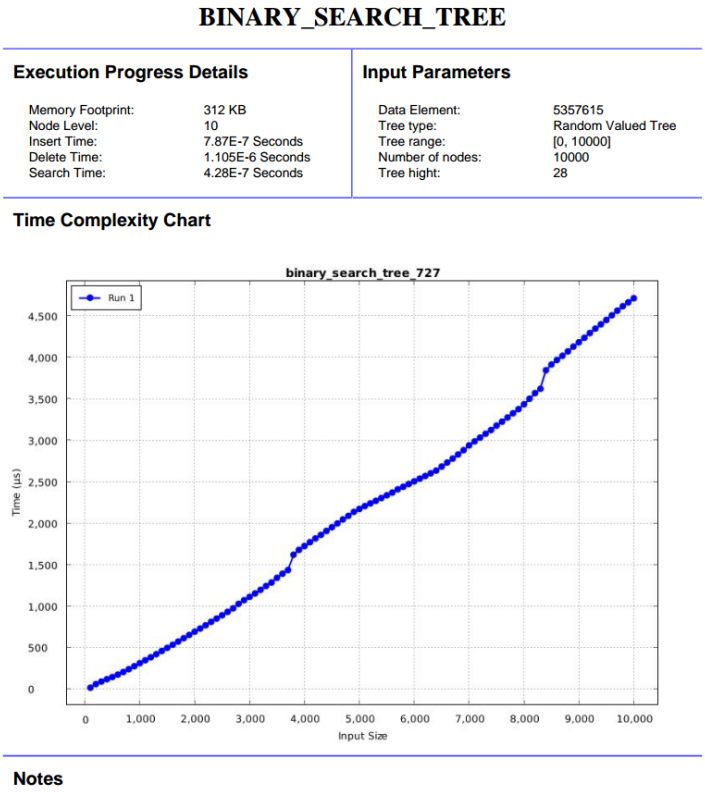


Figure 3.3: PDF for Binary Search Trees.

3.3.3 Editor and Adding notes to the result

As seen from Figure 3.3 the PDF documents contain the task parameters and the runtime chart, however, they do not provide any support for the user to include their observations and analyze results. This enhancement helps the Inf2B students express their thoughts easily and moreover allows them to start working on a document, save their report and go back to it later. This way they will be able to complete coursework exercises or just take a note for revision.

Why embedding an editor is not a practical choice

As discussed previously the Frontend of the workbench is implemented in Java. This specification would require finding an open source editor which is entirely written in Java. Moreover, since the aim is to provide Algobench for Linux, Mac and Windows users, any such editor should be available for all three platforms. One candidate meeting these requirements is JEdit [24]. However, after examining it's code, functionalities and building it from source [8] it became evident that it provides very basic features, which do not include working with pdf documents and converting between different file formats. Another major problem would be building the processes inside

Algobench Frontend for controlling the application and its functionalities. Furthermore, as described by Dave Wood, Marc Loy and Robert Eckstein in [34] most of the Swing components (the GUI toolkit Algobench uses) are light weight. Inserting a heavy weight components on top of a light weight contradicts with the Z-order of Swing components and leads to engineering problems. A different approach would be to build an editor from scratch. In `javax.swing.text` the class `JTextPane` [39] allows building a basic editor, however, a more complex functionality is desired. Therefore, this would require more time than the scope of this project and the outcome would not be guaranteed.

Embedding text editors in Java applications will need a lot of considerations, both legal and mostly engineering. It requires overcoming both platform and Java language challenges. This problem is open ended, very challenging and even more interesting. From my week of research on different engineering websites and platforms I discovered that there are not many people that have attempted to do it. Due to time limitations, I could not devote more time into researching about text editor embedding in Java desktop applications, however, this can be a suitable research task for a Masters or a PhD student who has more time than me. I approached the task in a different way. As suggested by my supervisor, a good compromise between functionality gain and user experience would be using an external editor.

Why LibreOffice is a good choice for an external editor

LibreOffice [33] is a good choice for an external editor first of all because it is open source and is available free of charge. Second, LibreOffice is provided for all three major platforms - Linux/Unix , Mac and Windows. Furthermore, it supports the ISO standardized Open Document Format (ODF), PDF import/export, Creation of Hybrid PDF and easy conversion between formats [17]. It's reach functionality allows the users to create and edit documents easily. It has engineering support, good documentation and stable releases. Last but not least LibreOffice is the text editor which is installed on the DICE Machines at the University of Edinburgh. All of the features described above make it good for the task and accessible to the Inf2B students.

Once a suitable editor was identified, building the functionality was the objective. After running a task the students are offered a choice of adding notes to the results. This option is only available if there are any results present, regardless if the task has been saved and loaded or just ran. After clicking the AddNotes icon if LibreOffice is not installed on the current machine a pop-up message, shown in Figure 3.4, is displayed at the front. If LibreOffice is installed, the `TaskMaster` object and the currently

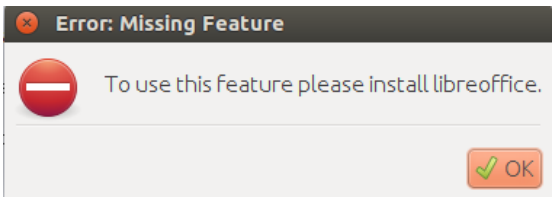


Figure 3.4: Error message.

selected task are used to identify the related results. Then the code checks for any pre-

vious PDF files related to the task that the user might have saved in the AlgoBench `/saved/` directory, which is located inside the tool's folder. This allows the students to keep working on a task they have previously saved and added notes to. If no related documents are found in the `/saved/` directory, the table with input/output details and the runtime chart generated from the XSL file are temporary saved as ODF file in the `/saved/` folder. The `Java Runtime Process` [23] starts here. The runtime process executes the ODF file into LibreOffice which opens on the screen. When finished working the user can export their changes into a PDF file which the editor offers to save in the `/saved/` directory. The ODF file that was previously used when firing the editor is deleted on exit. It is important to note that, for safety measures, at the rear event of a system crash, I have also implemented an "on beginning clean-up" which takes care of any temporary ODF documents left in the directory. These internal Java processes which are responsible for the editor, the conversion of the formats and the other operations related to this task are hidden to the user. Nevertheless, they required a synchronization with the other functionalities of the tool, such as running/rerunning a task, editing a task etc. With this the desired functionality was achieved.

3.3.4 Flexible runtime queries

The idea behind flexible runtime queries:
As mentioned previously, the inherited application provided the timing of whole algorithms and data structure operations (Table 3.1). The flexible runtime queries allow students to examine the time taken to execute code between two valid checkpoints. For the purpose of this report we define valid checkpoints to be any two checkpoints that enclose a computationally significant and meaningful code segment. For instance, the `merge` operation inside Merge sort, or the `partition` in Quick sort [36]. For algorithms which cannot be divided into phases (like mergesort) checkpoints can be placed around segments where the algorithm spends most time – for loops, while loops or recursive calls [41]. Going beyond the overall runtime of an algorithm, this feature allows the users to break the problem into segments and therefore it improves their understanding of runtime analysis. Furthermore, for better user experience, the pseudocode of the algorithm is displayed in the graphical user interface and the students are offered a choice of valid checkpoints. This way they can revise the code and use it to improve their understanding of the algorithm.
Algorithms suitable for checkpoins
Not all of the Algorithms and Data structures supported by the workbench (Table 3.1) are suitable for the flexible runtime queries. For instance, there are no meaningful checkpoint positions inside Binary search, apart from the initial timing mechanism. However, the `insert` operation inside Insertion sort is a meaningful segment of code [36]. Its timing changes depending on the order of the input array, therefore collecting the execution time can benefit the users' overall understanding of the runtime analysis. Table 3.2 revises the algorithms supported by Algobench and identifies possible checkpoint positions.
Merge sort and implementing flexible runtime queries
Due to the scope of this enhancement the main priority when implementing the functionality was building the data path from the Backend to the Frontend, along with the

Algorithm Group	Algorithm	Checkpoint
Sort	Quick sort	time for partition step
	Heap sort	time for building the heap time for removing the maximum element;
	Insertion sort	time for insertion
	External Merge sort	time for merging
	Internal Merge sort	time for merging
Graph	Breadth-First Search	no meaningful checkpoints
	Depth-First Search	no meaningful checkpoints
Hash	Hashing	no meaningful checkpoints
Search	Linear search	no meaningful checkpoints
	Binary search	no meaningful checkpoints
Tree	Binary Search Tree	time for insert time for search time for delete
Queue	Priority Queue	timing of PQ operations already provided

Table 3.2: Algorithms and Data structures suitable for flexible runtime queries.

mechanisms for setting and removing checkpoints. This means overcoming the challenges involved on both sides and giving an example for future developers by implementing the functionality for one of the algorithms discussed in Table 3.2. I proceeded with Merge sort.

After executing a Merge sort task, as previously the user is presented with the runtime chart of the algorithm. At this stage, buttons `Set Checkpoints` and `Remove Checkpoints` appear bellow the chart. The two buttons are currently only visible for Merge sort. They are synchronized appropriately, so setting checkpoints is only available to click if there are none set and removing them is available if the previous action has been performed. For a better user experience, the students are presented with the pseudo code, placed inside `JDialog` window (Figure 3.5). This pseudocode is taken from the inf2B study notes [26]. It is important to note that the three check boxes from Figure 3.5 are synchronized, thus no matter which one is clicked they are all selected. The functionality that synchronizes check boxes is provided at the Frontend, so if multiple checkpoints are possible, it would be easy for the future developers. The resulting chart is presented in Figure 3.6.

In order to achieve the functionality described above the data path of the tool was changed. Everything up to the request for execution at the Backend remained the same. The new implementations concern the Backend and the further data processing at the Frontend when results are returned to the GUI.

As mentioned in section 2.2 there are two IPC mechanisms – the anonymous pipes which connect the C++ standard output to the Frontend (used for the live updates) and the TCP socket used for requesting the execution and receiving the timings. Either of this can bring the checkpoint timings to the Frontend.

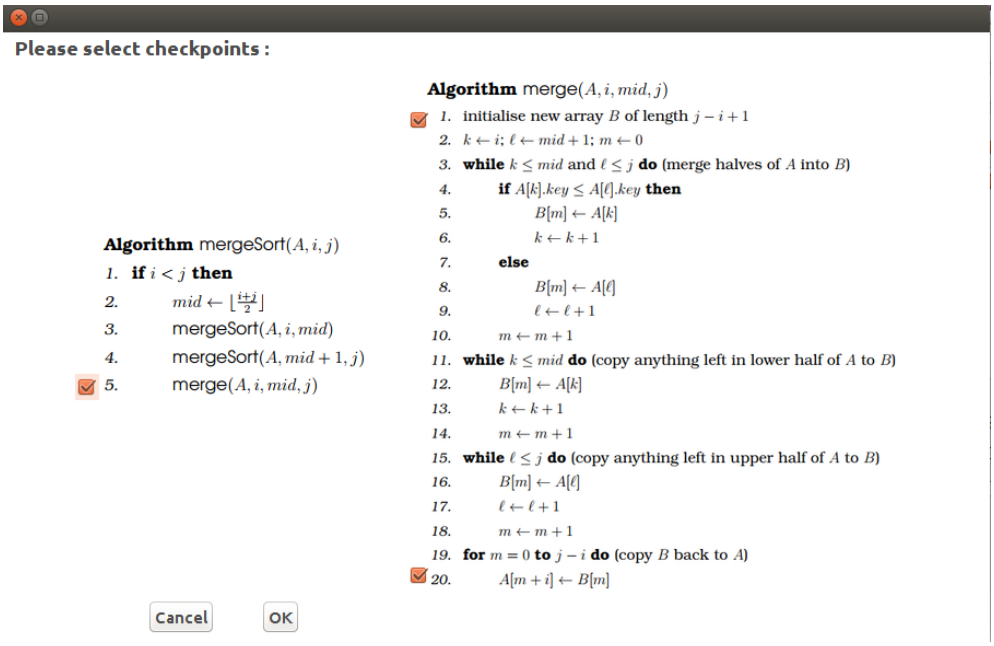


Figure 3.5: Flexible runtime queries. Checkpoints for Merge sort algorithm. Pseudocode is taken from the Inf2B notes [26].

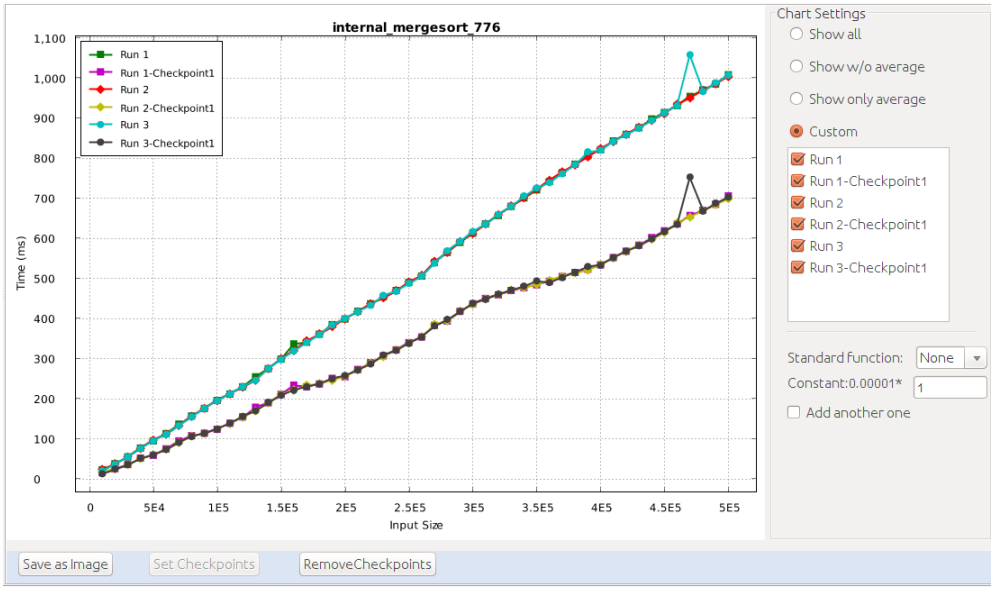


Figure 3.6: Checkpoints chart of Merge sort (3 runs). The upper three charts represent the timing of the whole algorithm. The bottom three charts are the cost of merging in each of the above graphs, respectively. Average values available from “Show All” selection.

Why the anonymous pipes are not a good approach

If we pick the anonymous pipes then a lot of functionality the tool already has can be reused. In this case going through the `Server` inside `TaskMaster` becomes unnecessary and the path of the data becomes significantly shorter. However, the anonymous pipes [4] were not created to handle big sequences of data, but only evanescent messages. They would not be able to relate the number of the checkpoint (in the general case more than one checkpoint per algorithm is possible) and moreover it would be hard to relate the data to its respective run. If a very strict ordering of the data is established and a lot of the parameters are being hardcoded at the Frontent this functionality would be possible, however, following good software engineering principals [48] it is more reasonable to use the TCP socket, even though it would extend the data path and would require more engineering work.

Why the TCP socket is a better approach

The data from the Backend is flushed through the socket in the form of String lines of tab separated values. The format is a single title line which defines the fields (input size, run1, run2 etc.) followed by the data in the same format. Essentially, at the Frontend, it is concatenated to a table containing the growing input size in the first column, followed by the timings. Passing the checkpoints' timings inside of this table would be more convenient than parsing unrelated outputs through the anonymous pipes. The new format received from the Backend is presented below.

```
Input size | Run1 |Checkpoint1-Run1 |  ... | CheckpointX-Run1 | Run2  | ...
```

This implementation is indifferent to the number of checkpoints and the number of runs. Furthermore, it is easy to relate the Checkpoints to the runs and identify the different checkpoints. Apart from the well formated data, another benefit of the TCP approach is the path through the `TaskMaster`. This way we can clearly modify the `TaskMaster` object to contain the checkpoints. The `TaskMaster` object related to a specific task is the object the workbench saves when the user decides to archive a task. By saving the checkpoints inside the `TaskMaster` object we handle the rear case when the users save a task into an archive, the next time they start the tool load the archive and set checkpoints before rerunning the task. If the data has not been inside of the saved `TaskMaster` object, it would have been lost and the tool would not be able to perform the functionality, which would ultimately lead to bugs. For the reasons mentioned above and to avoid this bug, I proceeded with an implementation using the TCP approach.

The implementation in the Backend involved collecting the timings of the merge operation and organizing the data in the format mentioned above. The overall timing of MergeSort [26] is computed in milliseconds. However, the deeper the recursion, the smaller the array gets. For refined results the checkpoints' timings were collected in microseconds, added and then converted into milliseconds for consistency.

After receiving the results at the Frontend I implemented a simple intermediate parsing which separates the runs from the checkpoints. The difficulties I encountered at the Fronthend were inside the `ResultsChartPanel` – the Swing panel responsible for showing the chart. After implementing the GUI (Figure 3.6) and its

corresponding class inside `Checkpoints.java`, there were two functionalities left to develop – adding checkpoints and removing them. None of them is trivial and they differ significantly from what has been implemented previously. To obtain the graph in Figure 3.6 and have the legend and the graphs' list (inside the Custom checkbox window) in this order, there are four objects which have to come together into two. The checkpoints and the runs are contained inside two maps with entry set signature `Map.Entry<String, XYSeries>` where the name of the series is Run# or Checkpoint#-Run# and the XYSeries is the data related to the name. There are two such maps, one for the checkpoints and one for the runs, they come together into one map and `JCheckBox` list which contains the checkbox list on the right of the screen. It is important to note that the checkpoints map is virtual and is never used purely for visualization, but only as a container. Iterating over the maps directly and altering order and entries in Java is considered to be a fail-fast iteration [38], which rises `ConcurrentModificationException` [18]. This exception occurs at runtime, it is unpredictable and the system cannot warn the developer at compile time. To walk around it and to achieve the sequential ordering (Figure 3.6), I cloned the data from the two maps into four vectors, joined them carefully into one map and a single `JCheckBox` list, following the special order seen in figure 3.6. I revalidated the series discussed above along with the ChartHolderPanel (the main container) and repainted the graph. Having done that, removing the checkpoints was an operation mirroring the approach described above.

3.3.5 Updating the chart package

Updating the workbench with the new `org.knowm.xchart` [29] library and removing the deprecated `com.xeiam.xchart` [50] package has been postponed since the first development iteration. As identified by Yufen Wang [49] replacing the entire library is a very much need update, however, due to time limitations she could not do it. Shalom Rachapudi [41] identified the same problem in his Masters Thesis, saying that this enhancement must be treated as major functionality as it is complex and will take a substantial amount of time. Due to time limitation he could not update Algobench either.

By this development iteration the workbench has grown to be a very complex system, containing more than 30,000 lines of code, most of them written in Java. A lot of the features provided by the tool operate with the charts – the users can view a chart, compare it, save the graph and now even create appropriate PDFs and add checkpoints. With a tool growing that rapidly the more we wait to update such a major component as the chart package, the harder and more infeasible this task becomes. Using the deprecated `com.xeiam.xchart` [50] package mean that there is no support for the toolkit. Furthermore, we are depriving ourselves from all the new, rich and useful functionality of the `org.knowm.xchart` [29] library (described in 3.2). Updating the chart library is complex, once the replacing with the new package starts, debugging becomes very hard since the other components using the old library fail to work. Due to the scope of this enhancement in the following paragraphs I am going to discuss mostly the high-level changes. Without any further discussions, let's dive into the "implementation".

Algobench operates using two types of charts, bar charts for Hashing tasks and

Chart Type	Builder	Styler	Series	Allowed Data Types	Style
XYChart	XYChartBuilder	XYStyler	XYSeries	Number, Date	Line
CategoryChart	CategoryChartBuilder	CategoryStyler	CategorySeries	Number, Date, String	Bar

Table 3.3: XYChart and CategoryChart (used by Algobench) and their related classes [30].

Line charts for all the rest. In the old package to build a specific chart one should pass an `enum` type, namely `StyleManager.ChartType`, into the `ChartBuilder` constructor. However, in the new library instead of a single `ChartBuilder` and a single `StyleManager`, every chart type is provided with a separate `Builder` and `Styler` class. The deprecated `com.xeiam.xchart` [50] library also used a single `Series` class, along with an `enum` defining the type. In the new library every chart type is provided with a separate `Series` type class. Table 3.3 contains the respective `Builder`, `Styler` and `Series` for the Line and the Bar charts, namely `XYChart` and `CategoryChart`. This high level replacement indubitably allows more functionality and customization for the charts, however, it brings issues for the workbench. To update the workbench the plotting mechanism had to change, the two chart types could no longer be extended by one class. In Algobench plotting is possible with the help of the `Plotter` class [47, 49]. Its most important functionality is to parse the results from the backend and create the appropriate chart (bar or line) by using `MyChart` class, a custom class written in the second development iteration [49]. Due to the changes between the old and the new library `MyChart.java` could no longer be used for both bar and line charts. After fixing the Hashing bug at the backend, described in section 3.3.2, I started working on the new bar chart functionality. As mentioned before the new library provides some very complex functionality. These features are more than enough to support the operations the workbench applies on bar charts (`CategoryChart`). For this reason, I did not implement a special custom class and a builder class for `CategoryChart` but instead provided the new functionality of creating a bar chart in a `getBarChart()` method, shown in Figure 3.7 . The code creates the chart using the corresponding `Builder`, adds the data to the `Series` and `Styles` the appearance using the appropriate API [28]. The line charts (XYChart) were a rather more complex case. They involve the functionality of comparing tasks, adding checkpoints, adding standard functions to the graphs (log, NlogN etc.), creating PDFs, saving the chart, removing and adding Runs from the chart. All of these involve operations with the series, which need customization. For this reason new `LineChart` and `LineChartBuilder` classes which extend `XYChart` and `XYChartBuilder` from the new library were created. The class `LineChart` contains methods invoked inside the `ResultsChartPanel` when the user adds/removes a standard function or selects to view only specific runs. The new `LineChartBuilder` overwrites the methods from its parent class. Table 3.4 presents an overview of the methods inside the class `LineChartBuilder`. It is important to note that the new classes do not change the functionality of the workbench, which was developed in the last three development cycles [47, 49, 41] and in this development iteration by me. They accommodate the tools needed for it, using the new `org.knowm.xchart` library. After the changes described above the line chart could be easily returned from the `Plotter` class in a similar way to the bar chart. Even though with these changes plotting the results into the `resultsChartPanel` has remained simple and similar to

```
public CategoryChart getBarChart() {
    // create chart
    CategoryChart chart = new CategoryChartBuilder().width(800).
                                                     height(600).
                                                     title(title).
                                                     xAxisTitle(xAxisLabel).
                                                     yAxisTitle(yAxisLabel).
                                                     build();

    for (int i = 0; i < ySeries.length; ++i) {
        if (xData.length > 0 && ySeries[i].length == xData.length) {
            chart.addSeries(seriesTitles[i + 1], xData, ySeries[i]);
        }
    }
    // Customize Chart
    chart.getStyler().setLegendPosition(Styler.LegendPosition.InsideNW);
    chart.getStyler().setHasAnnotations(true);

    return chart;
}
```

Figure 3.7: getBarChart method inside Plotter class. Used to obtain the Hash (Bar) chart. Returns a CategoryChart.

before:

```
p = new Plotter(this.task.getTaskID(), String.valueOf(response), 0);
resultChartPanel = new ResultsChartPanel(task.getTaskID(), task);
resultChartPanel.addResultChart(p.getLineChart());
```

Changes due to the package replacement were found across 4 more files, due to using methods of the `LineChart's` super class, namely XYChart. However, after the plotting mechanism was done, fixing these errors was simply a matter of rigorously reading the new API [28]. It is important to note that the new library brings changes on many levels, from using separate classes for different charts, their builders, stylers and series (Table 3.3) to data types inside some methods . For instance, when adding series instead of type `Collection<? extends Number>` it uses `List<? extends Number>` or instead of keeping the data inside objects (Double, Integer etc.) it saves it in a primitive format (double, int etc.). As described by [14] collections are designed to be general and lists to be fast. These involved work-around different exceptions in the development process. Nevertheless, the new `org.knowm.xchart` package provides efficiency and features at the same time. The workbench was updated entirely. Future developers will thus be able to experiment with these new features and implement them into the workbench.

Method name	**Input**	**Output**
addSeries	String name, XYSeries series	XYSeries
addSeries	String name, double[] xData, double[] yData	XYSeries
addSeries	String seriesName, List<?>xData, List<?>yData	XYSeries
showSeries	String name	boolean
hideSeries	String name	boolean
showOnlyAverageSeries	None	boolean
showWithoutAverageSeries	None	boolean
showAllSeries	None	boolean

Table 3.4: Some of the methods from LineChart class.

Chapter 4

Testing and Evaluation

This chapter discusses the Mergesort algorithm and the Priority Queue data structure. These algorithms have been chosen as illustrative examples of the work carried out in this development iteration. Mergesort is used to present the checkpoints setup, while Priority Queue is used to show the PDF generation and the Adding Notes functionalities. These are the two tasks performed as part of the user evaluation of the workbench. Results from the evaluation, and testing methods during the development process are discussed at the end of the chapter. Other pre-existing features of the workbench have been given user trials by the previous developers. Some of the past and the current responses which bare similarity are discussed at the end of Section 4.4.1.

4.1 Internal Mergesort Algorithm

This section presents the process of the Internal Mergesort Algorithm from creating the task to executing it and viewing the chart. It is important to note that all of the algorithms Algobench supports are similar in the setup process, however, with different details. More attention has been given to the features developed in this development cycle.

Figure 4.1 presents the initial set-up panel. At this stage the users can configure the algorithmic group, the respective algorithm and the name of the task they want to execute. If no name is provided then the workbench concatenates the name of the algorithm with a random number to create a unique task name. Figure 4.2 presents an overview of the run, while Figures 4.3 and 4.4 show the resulting chart and the table. Figure 4.5 and 4.6 demonstrate the checkpoints set-up.

4.2 Priority Queue data structure

As mentioned earlier, Priority Queues were developed in the current development iteration. The implementation of the data structure was carried out using Heaps, however, other structures can be used to implement priority queues. Figures 4.7 and 4.8 present the settings which can be configured by the user. Figure 4.9 contains the results chart after executing the task, while in Figure 4.10 presents the functionality of adding notes to the PDF report generated by the workbench.

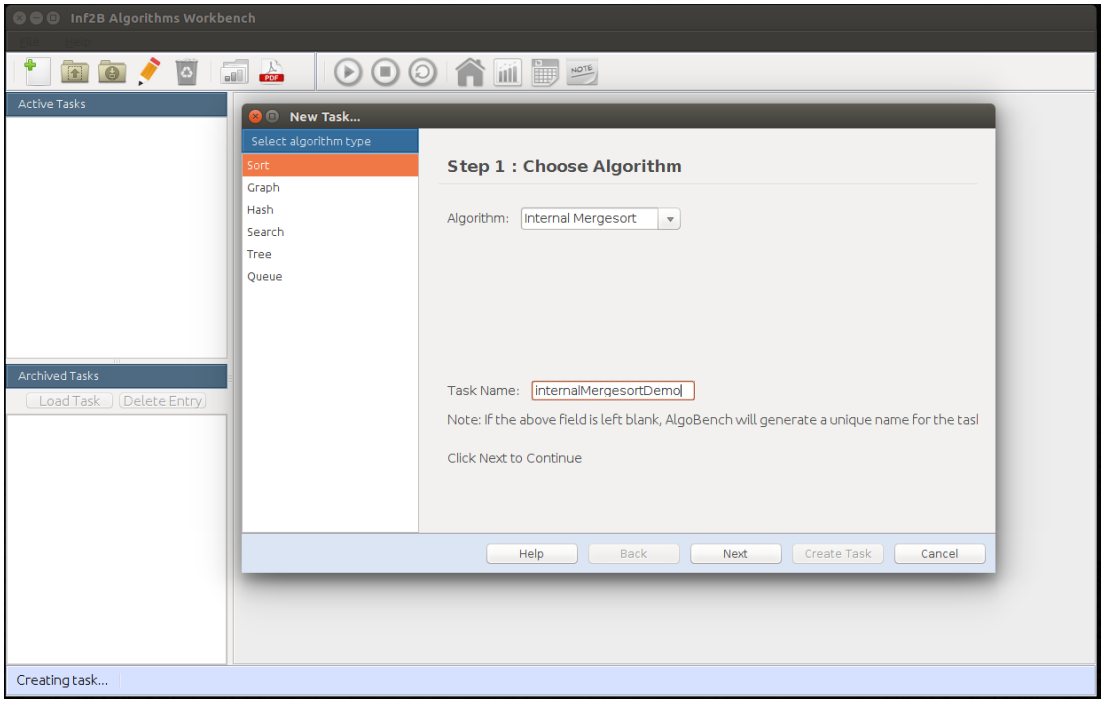


Figure 4.1: Creating Mergesort task.

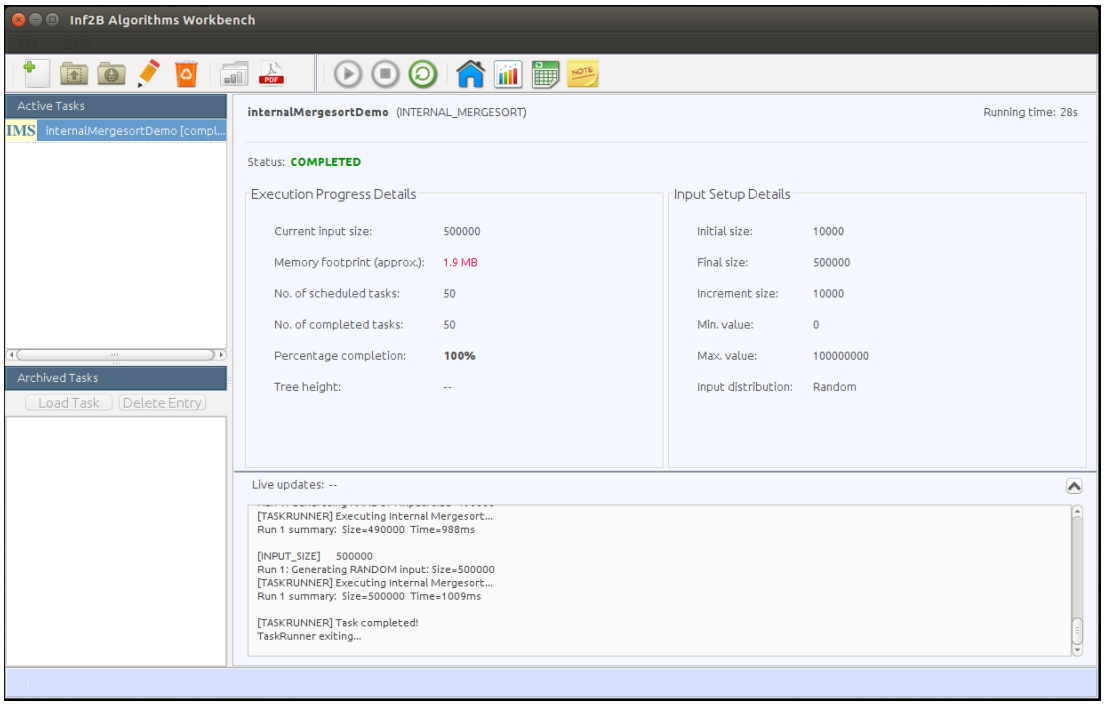


Figure 4.2: Overview of the executed Mergesort algorithm.

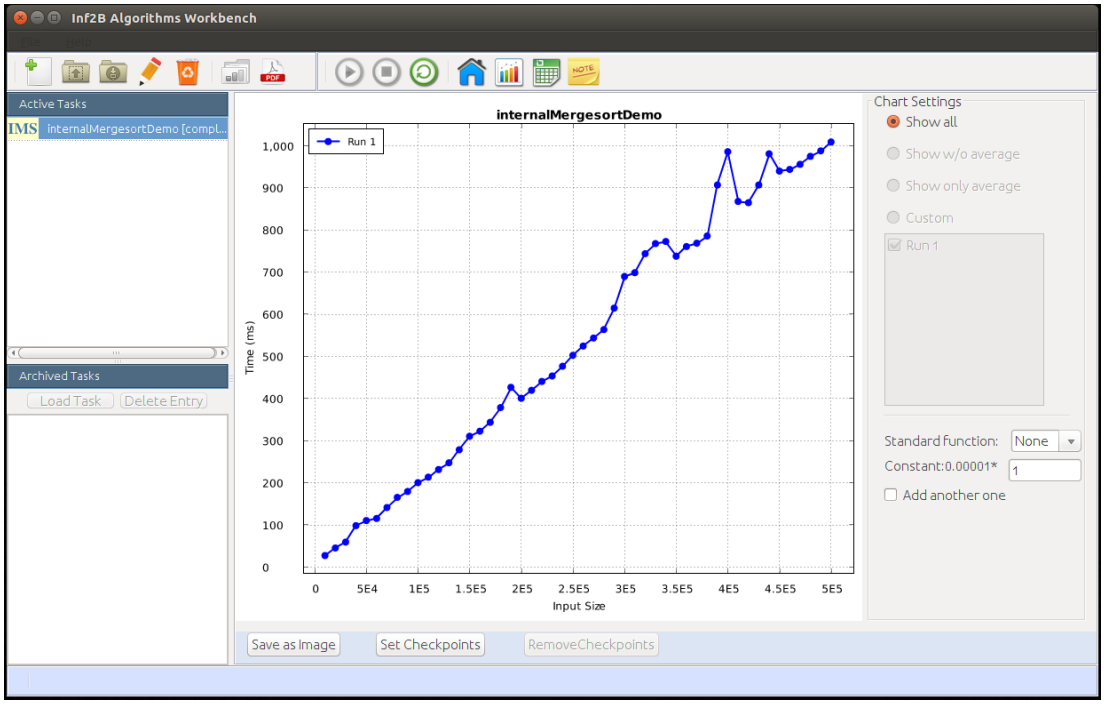


Figure 4.3: Meregesort results chart.

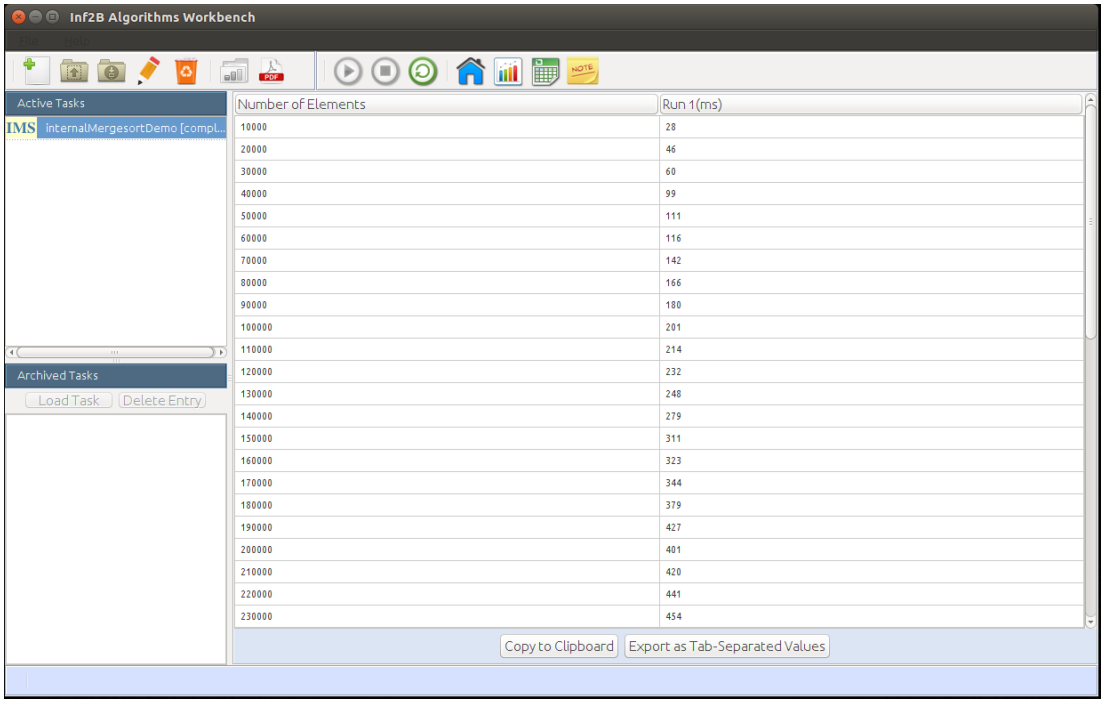


Figure 4.4: Results table and .csv export.

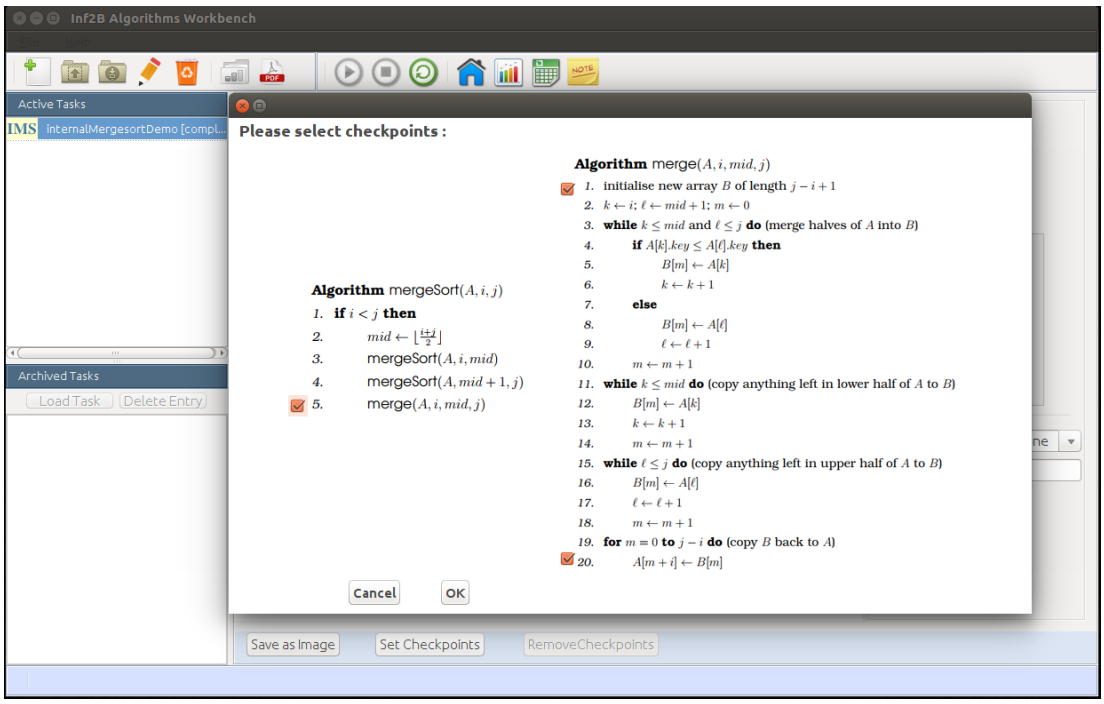


Figure 4.5: Setting checkpoints to the result (pseudocode obtained from [26]).

4.3 Testing during development

The workbench does not have implemented unit tests, so during the development process testing was mostly done by debugging, assertions and helpful prints. As stated earlier, making design decisions was done considering possible bugs which may arise in order for the most safe and beneficial approach to be adopted. Along with this at about 3/4 of the development cycle I asked another developer to test the software by exploring the functionalities of the Workbench from a user's perspective. He was not involved in the development process which made him a good candidate for testing in an environment closely mirroring production. This helped in catching the Hashing bug (discussed in Section 3.3.2), which I later tracked down in the software and fixed.

4.4 User Evaluation

The workbench was evaluated by current Informatics students from the University of Edinburgh. The evaluation format and questionnaire used in this development cycle were originally proposed and used by Eziama E. Ubachukwu [47] and later used by Shalom Rachapudi [41]. The format makes use of Likert scales. For the convenience of the reader, evaluation tags are discussed below. The tags are based on the definitions mentioned in [47, 41].

- *Likert scale L1*: "Very poor", "Poor". "Neutral", "Good" and "Very good" are used to express level of agreement. The adjective used varies from task to task. For example, in tasks asking how easy or difficult the action is, "easy" and "difficult" are used.

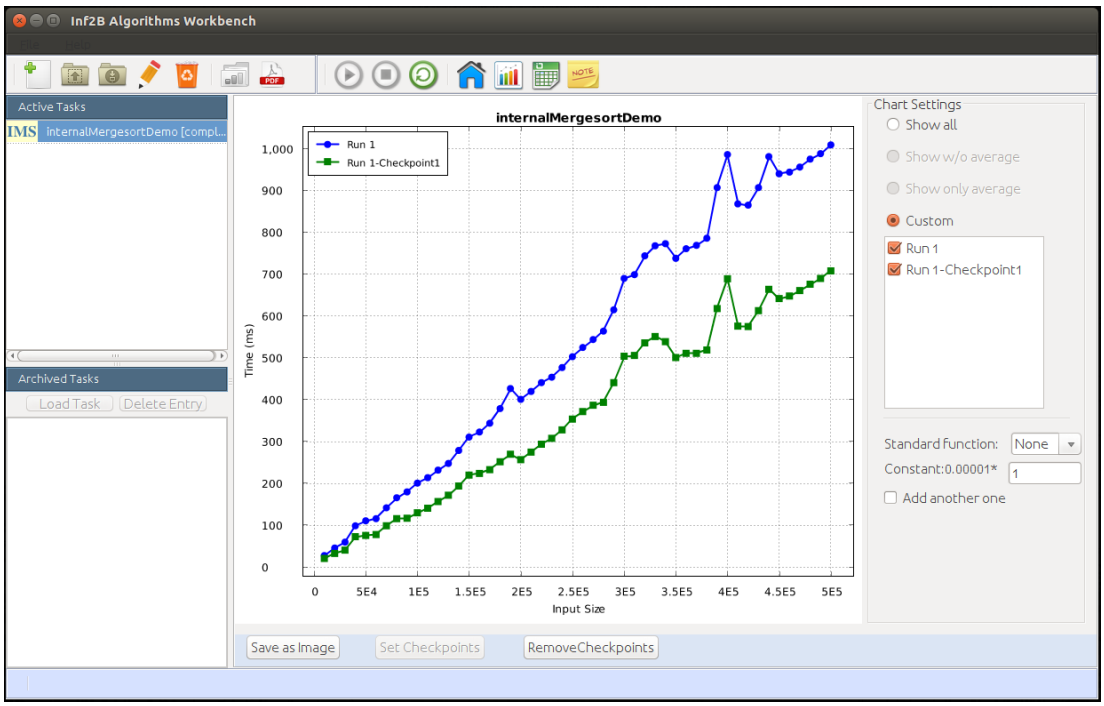


Figure 4.6: Runtime chart of Mergesort and runtime chart of `merge` method.

- *Likert scale L2*: "Yes", "No", and "Can't' tell".

- *Text*: Type of evaluation allowing suggestions, open comments and critique.

User evaluation is the best way to see how much the workbench fulfills the needs of the users. The key factors being evaluated were user experience and satisfaction, easy-to-use and functionality set of the tool. It is important to note that all of the previous developers [47, 49, 41] have been Masters students and past AlgoBench trials have been conducted during the summer, a time when the Inf2B students are away. This is the first user evaluation which involved predominantly Inf2B students, making the collected responses particularly important for the objective assessment of the workbench.

A total of 9 students took part in the evaluation, 7 of whom were current inf2B students. They were divided into groups based on their academic year. Table 4.1 presents the distribution of the participants. The participants were treated anonymously and apart form their current academic year they were not identified in any other way. To achieve unbiased, impartial and correct responses, none of the participants had previously met the developer conducting this study (me). They were left to explore the tool by completing the two tasks mentioned before, without any more guidance. The evaluation was conducted in a drop-in basis, so I was also present in the room. However, that was only to answer questions and to make sure that the participants could complete the task if they needed help.

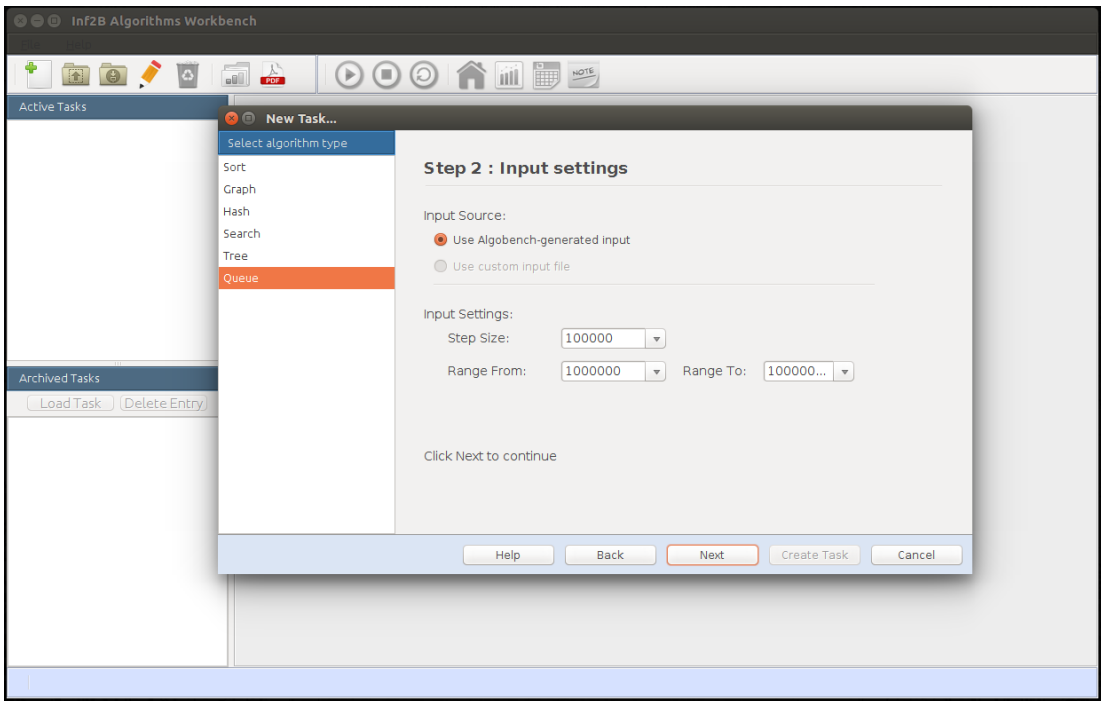


Figure 4.7: Setting Step and Range of a Priority Queue task.

Class	Number of Participants
UG2	7
UG4	2

Table 4.1: Participants distribution

4.4.1 Evaluation Results

The full evaluation report can be found in Appendix A. The first of the two tasks asked the students to run an Internal Mergesort algorithm with default settings and then after the execution to set checkpoints to the result. Among the 9 students 66.7% found carrying out the instructions to be *very easy*, 22.2% *easy*, while the rest 11.1% said it was *neither easy nor difficult*. Also 88.9% of the participants said that the Mergesort graph showed the expected trend and the rest choose *can't tell*. The trend on the checkpoints graph was a bit more challenging, however, 77.8% identified it correctly, while the rest *could not tell*.

The second task asked the students to create a priority queue with default task name and insert the worst case data element. The input range had to be between 1,000,000 and 10,000,000 with step size of 100,000. When finished with the execution and viewing the graph, the students had to add notes to the experiment. For this task, 66.7% found the instructions to be *very easy*, while 22.2% found it *easy*, the rest voted for *neither easy nor difficult*. 77.8% said that the graph indeed is correct, while the rest *could not tell*. The process of adding notes to the experiment was *neither easy nor difficult* for 11.1% of the participants, while the rest voted for *easy* and *very easy*.

- **Functionality:** 77.8% voted for *very good* and 22.2% voted *good*

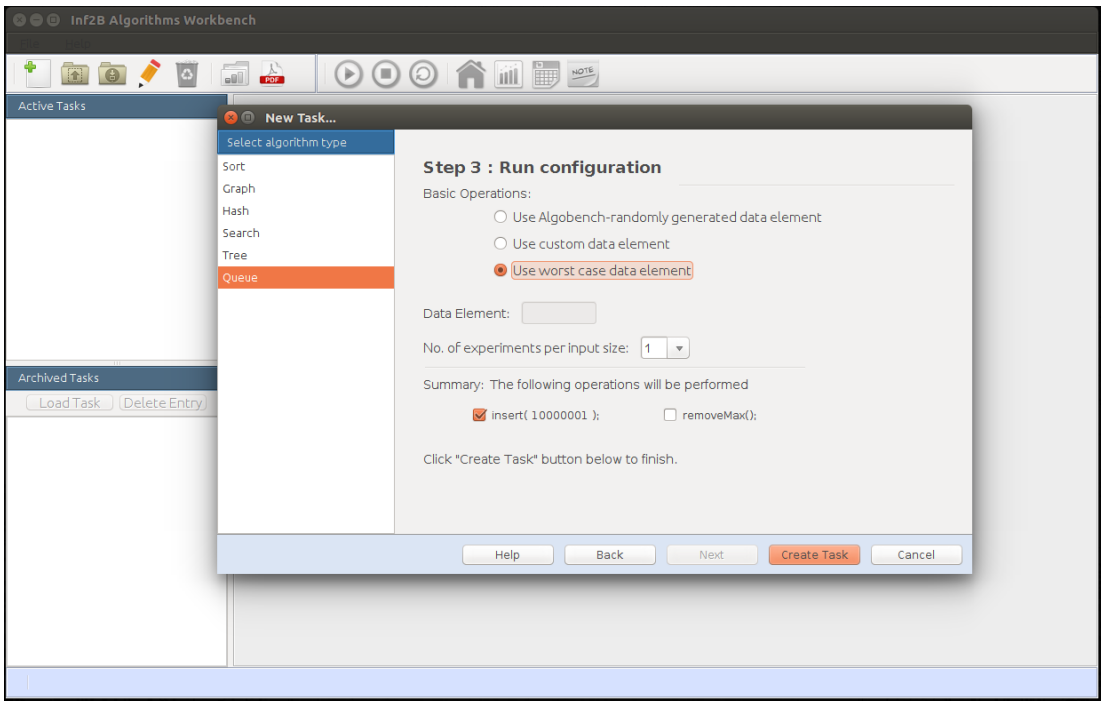


Figure 4.8: Setting the desired Priority Queue operation.

- **User Interface Design:** 33.3% voted for *very good*, 44.4% voted *good* and 22.2% chose *neutral*
- **Easy to use:** 77.8% voted for *very good* and 22.2% voted *good*
- **Live updates from the Backend:** 55.6% voted for *very good*, 22.2% voted *good* and 22.2% chose *neutral*
- **Features:** 55.6% voted for *very good*, 33.3% voted *good* and 11.1% chose *neutral*
- **How useful this tool will be for the students:** 77.8% voted for *very useful* and 22.2% voted *moderately useful*

In essence, the new version of the workbench was well accepted by the Inf2B students. They found it easy to use and liked the provided features and functionalities. Most of them judged the tool to be useful in the learning process and made some good suggestions about the future development. In the free text comments they propose more animations, graphs and colour improvements. It is important to note that in the previous trial, carried out by Shalom Rachapudi [41], some of the participants made similar comments. I have considered all the suggestions and included some possible enhancements in Section 5.1, which are going to improve the user experience. Altogether, the evaluation assessed the tool, it's new features and gave a good insight into the Inf2B students opinion of the tool.

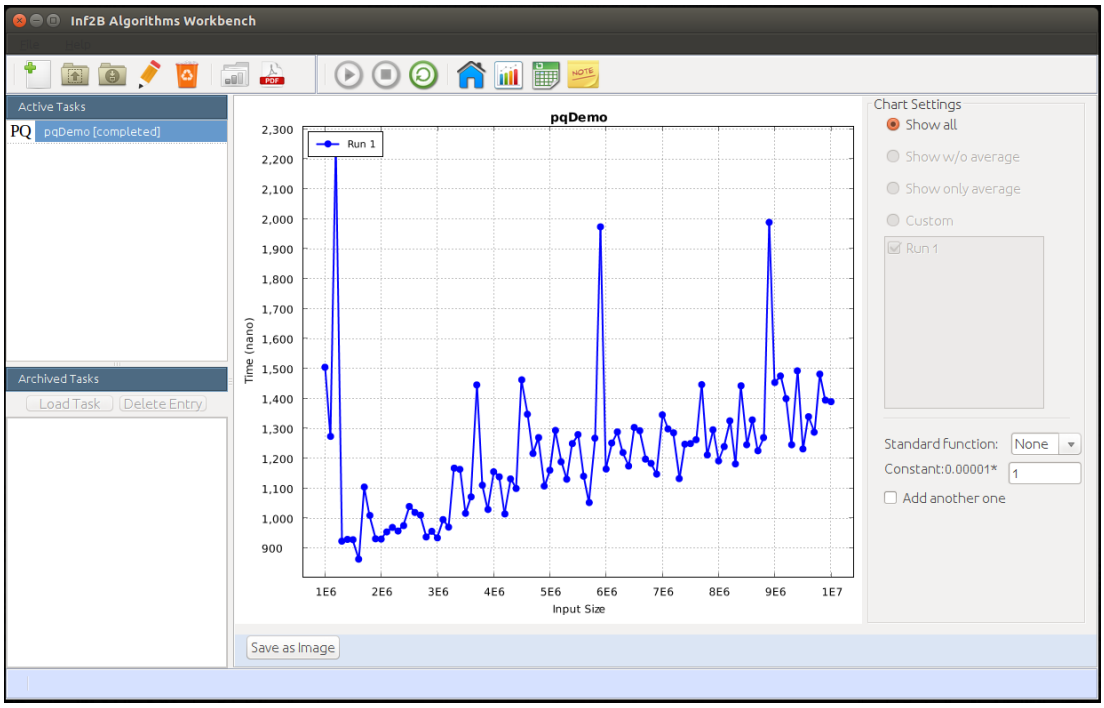


Figure 4.9: Priority Queue chart.

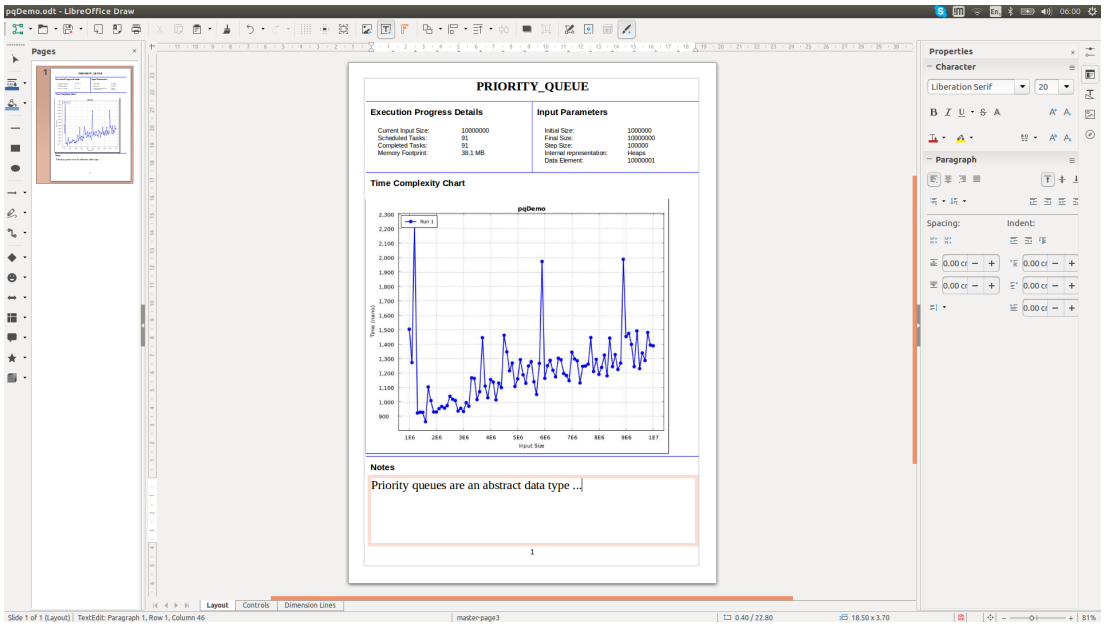


Figure 4.10: Adding notes to the result. The PDF report inside LibreOffice [17].

Chapter 5

Conclusion and feature work

In this report, the design and implementation of new features and functionalities enhancing a unique tool, named AlgoBench, have been discussed. As mentioned in Chapter 2 there are many tools that aim to visualize algorithms and data structures. However, tools designed to examine time and space complexity, like AlgoBench, are a much rearer find.

The background and the rationale behind the choice of C++ and Java languages, the Inter-Process Communication (IPC) mechanism and other main components of the Workbench have been discussed in Chapter 2. The contributions of the previous developers Eziama E. Ubachukwu [47], Yufen Wang [49] and Shalom Rachapudi [41] were presented in Section 3.1, alongside the educational value of these achievements.

To put my contributions to this project in a nut shell:

- The old and deprecated chart package, used for plotting, was replaced by a new library with much richer functionality.
- The Workbench was enhanced to support flexible runtime queries between two checkpoints.
- PDF export, originally started by Shalom Rachapudi [41] was further developed and finalized by me.
- AlgoBench has also been extended to support adding notes to the PDF documents.
- Priority queue data structure was added to the workbench.

In the development process a few small bugs have been identified and fixed. With this all tasks originally planed were completed.

The user evaluation of the Workbench, conducted with the Inf2B students has given invaluable insight into what they anticipate from the project. Even though the sample size was small, some very useful suggestions have been provided for the future developers to contemplate when developing AlgoBench further.

Overall, this has been a challenging, exciting and engaging project. Contributing to real users is very important to me and has made me enjoy every bit of the development process. Please note that even though this is the end of my Honours project, I am very much excited about the future of the project and would continue to contribute to AlgoBench whenever time permits.

5.1 Possible Enhancements of the Workbench

In this section I will discuss possible enhancements to the workbench. If not otherwise specified, the following proposals are originally my ideas.

Further development on the Checkpoints

It would be useful if this feature could be provided for more of the suitable algorithms shown in Table 3.2. Moreover, a nice future enhancement would be to develop further the functionality of removing checkpoints. Currently after adding checkpoints to the graph the student can remove all of them at once, I suggest using an additional `JDialog` window for displaying the pseudocode again and giving them the opportunity to pick specific checkpoints they want to remove. These two enhancement will both enrich functionality and improve usability.

Improving user experience by adding new features

Some of the free text responses from this and the previous evaluations [41] of the workbench suggested that it would be good if the tool provides more animations, graphics and colorful and interactive features. Improving user experience with the workbench could be a stand alone project for a future developer. One useful enhancement would be adding Real time charts. The newly updated `org.knowm.xchart` package provides support for this functionality [29], thus the future developers can make use of it. This enhancement will improve the design by providing interactive charts and presenting useful information in real time. Another good feature to add would be showing the standard function ($\log N$, $N \log N$ etc.) along with the results. Currently after running a task and viewing the graph, the students can fit a function into it by experimenting with a set of predefined functions and free input constant for multiplication. It would be much easier for the students if they can request the tool to do that for them. It is important to note that one can argue for and against this enhancement, however, since it was suggested by some of the Inf2B students who took part in the evaluation I present it here for future consideration. These functionalities along with small improvements like colour scheme, suggested at previous user evaluation of the workbench, will lead to a more attractive interactive experience for the users.

Algorithm and Data structures Visualization

First proposed by Eziama E. Ubachukwu [47] this enhancement will be a good extension to the workbench. Visual learning is one of the most common styles of learning. Currently the focus of the workbench is to provide the runtime charts. However, it would be useful to visualize some of the data structures like priority queues and binary search trees, respectively with their operations. By seeing the steps of an operation executed on the graph the students will be able to both improve their knowledge of the algorithm and understanding of the runtime analysis. This is a major enhancement which will help the learning process and will transform AlgoBench into a more robust pedagogical tool [49]. As mentioned in Section 2.4 the ADV [10] tool provides support for visualizations. One possible approach would be to integrate the ADV tool with

AlgoBench. This would be a major undertaking but would avoid duplicating a lot of work.

Adding more Algorithms and Data Structures

After this development iteration the Workbench supports a total of twelve algorithms and data structures. All of the past developers [47, 49, 41], including myself agree that extending this set should be a top priority for the future development. A possible data structure, taught in the Inf2B course, but not yet supported by the workbench is AVL Trees including the operations to `insert`, `search` and `delete` an element. Providing support for it will be a good starting task for next developers. Moreover, it would be interesting to show how different implementations of data structures change the performance of their operations. For instance, implementing priority queues with binary heaps gives $N \log N$ worst case runtime for inserting element and removing the element with maximum priority. However, if an alternative implementation is made using linked lists insertion becomes at most N, while removing the element with maximum priority becomes constant. Comparing the two implementations, the users will develop better understanding of them, the runtime and the possible benefits of either of them.

AlgoBench as a web service

This enhancement was originally proposed by Shalom Rachapudi [41] at the end of the third development iteration. Even though the first steps towards achieving this goal will be made by a Masters student during this summer I find it important to emphasize the importance of this task. This is a very complex task which will involve a lot of possible issues like security, memory issues, scalability etc. It will most likely require a few development iterations to complete. However, achieving this objective will be an enormous achievement because it will make the workbench more accessible to both the Inf2B students and to the other learners.

Appendix A

User Evaluation Report

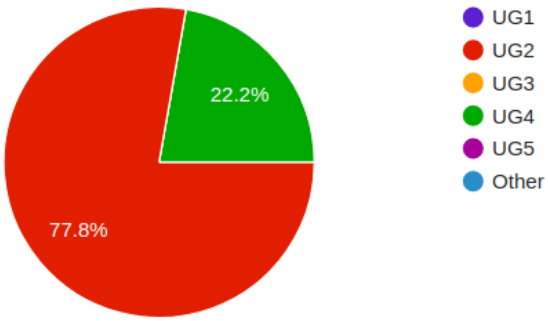
The evaluation format of this questionnaire was originally proposed and used by Eziama E. Ubachukwu [47] and late used by Shalom Rachapudi [41]. The following document presents the results from the user evaluation carried out by me in this development cycle.

AlgoBench User Evaluation

9 responses

What class are you about to finish in 2017/2018 academic year?

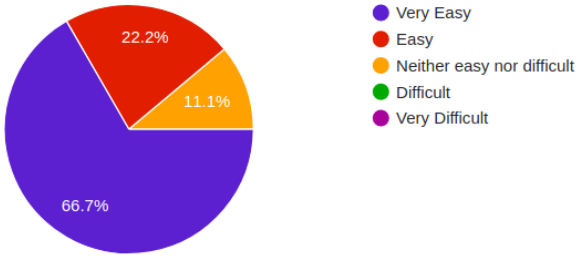
9 responses



Task 1: Internal mergesort and Checkpoints

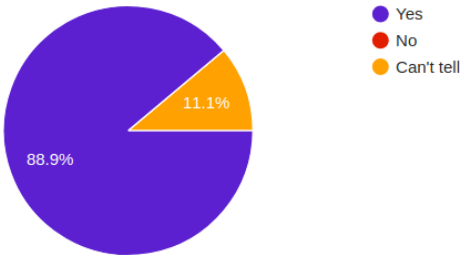
How did you find running the given task, viewing the runtime graph and setting checkpoints to the result?

9 responses



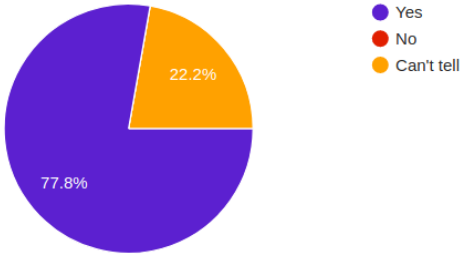
Did the runtime graph show the expected trend?

9 responses



Did the checkpoints graph show the expected trend?

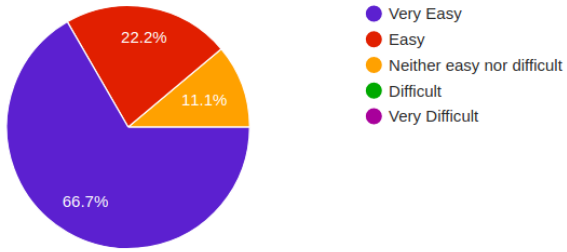
9 responses



Task 2: Priority Queue and Adding Notes

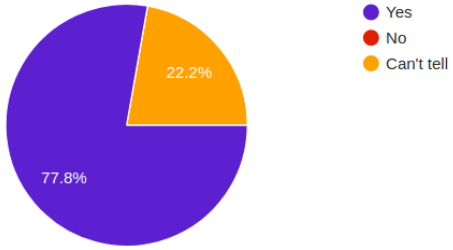
How did you find running the given task and viewing the runtime graph?

9 responses



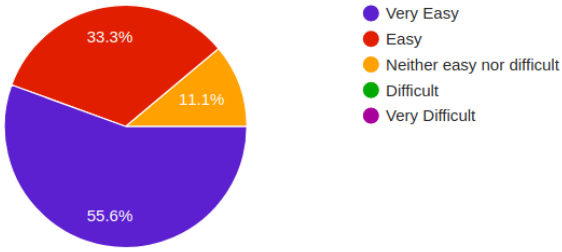
Did the graph show the expected trend?

9 responses



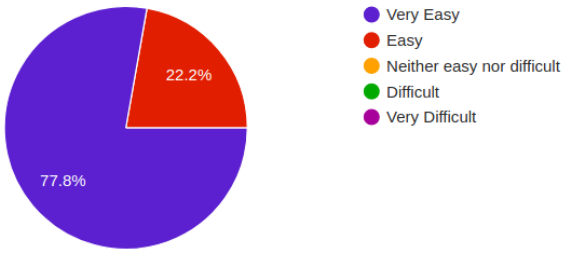
How did you find adding notes to the experiment?

9 responses



How did you find carrying out the instructions?

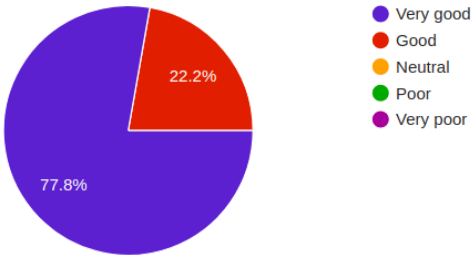
9 responses



Overall criteria

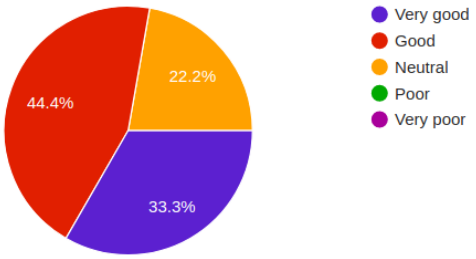
Functionality (ability to do what it was designed for)

9 responses



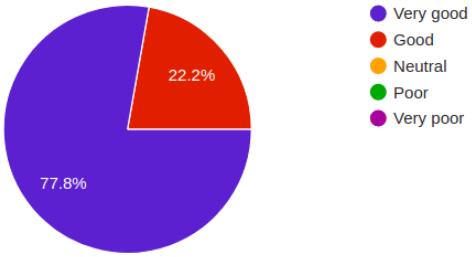
User Interface Design

9 responses



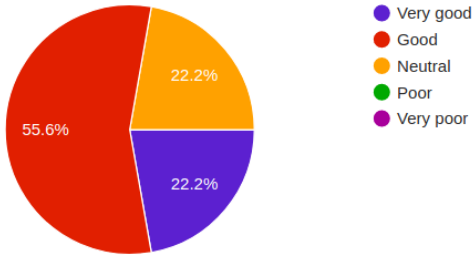
Easy to use

9 responses



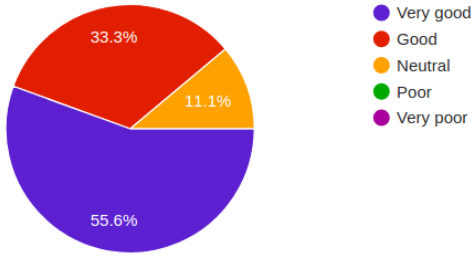
Live updates from the backend

9 responses



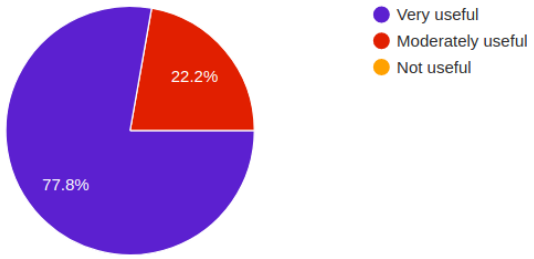
Features (PDF generation, adding notes to results, checkpoints etc.)

9 responses



How useful this tool will be for students learning Algorithms and data structures in inf2B course?

9 responses



Comments

Are there any features or functionality you would like added or improved?

5 responses

I liked the tool and the notes option, can't think of anything to change. Maybe showing the expected graph along with the results.

UI: good that it has percent completion when running, but number scheduled/completed tasks would be nice to be in X/Y format. I think the buttons at the top should be visibly clickable.

I am not good at Mathematics, so it would be helpfull if the expected trend function is shown with the graph. But apart from that it will be very useful for the class.

Testing the tool was interesting, maybe adding some animations and making the tool more colourful would improve the interaction

The live updates were printed very fast which was a bit confusing. I liked the editor option and the graphs. The application seems useful for students without much experience with algorithms.

Bibliography

[1] Sorting - a visualization of the most famous sorting algorithms. [Accessed 20-March-2018]. [Online]. Available: http://sorting.at/

[2] Xchart - a simple charting library for java. [Accessed 20-March-2018]. [Online]. Available: https://knowm.org/open-source/xchart/

[3] Xchart - xeiam. [Accessed 20-March-2018]. [Online]. Available: http://javadox.com/com.xeiam.xchart/xchart/2.3.2/com/xeiam/xchart/package-summary.html

[4] D. Ahuja. (2003) Unleashing anonymous pipes part 1. [Accessed 7-April-2018]. [Online]. Available: https://www.codeproject.com/Articles/4816/Unleashing-anonymous-pipes-Part

[5] F. Alhosban and M. Hamad, "The effectiveness of aural instructions with visualisations in e-learning environments," Ph.D. dissertation, Durham University, 2011.

[6] Apache^{TM}. Apache^{TM} fop: Embedding, 2017. [Accessed 23-March-2018]. [Online]. Available: https://xmlgraphics.apache.org/fop/2.2/embedding.html

[7] Apache. Apache fop: Embedding. [Accessed 1-April-2018]. [Online]. Available: https://xmlgraphics.apache.org/fop/2.2/embedding.html

[8] A. Balaam. Building jedit from source. [Accessed 7-April-2018]. [Online]. Available: https://www.artificialworlds.net/blog/2012/02/12/building-jedit-from-source/

[9] P. E. Black, "Dictionary of algorithms and data structures. national institute of standards and technology," 2015.

[10] C. Cadzuwa and Y. Shao. Jedit - programmer's text editor. [Accessed 3-April-2018]. [Online]. Available: https://www.inf.ed.ac.uk/teaching/courses/inf2b/resources/adsSoftware.html

[11] K. Chansilp and R. Oliver, "Reusable and shareable learning objects supporting students learning of data structures in university courses," 2006.

[12] CMake. Cmake–build, test and package your software with cmake. [Accessed 7-April-2018]. [Online]. Available: https://cmake.org/

[13] cppreference.com. Constructors and member initializer lists. [Accessed 1-April-2018]. [Online]. Available: http://en.cppreference.com/w/cpp/language/initializer_list

[14] K. Cwalina and B. Abrams, *Framework design guidelines: conventions, idioms, and patterns for reusable. net libraries*. Pearson Education, 2008.

[15] M. K. Dalheimer, "Qt vs. java a comparison of qt and java for largescale, industrialstrength gui development," *Klarälvdalens Datakonsult AB kalle@ klaralvdalens-datakonsult. se*, 2010.

[16] G. David. Data structure visualization, 2011. [Accessed 20-March-2018]. [Online]. Available: https://www.cs.usfca.edu/galles/visualization/about.html

[17] DocumentFoundation, "Feature comparison: Libreoffice - microsoft office," [Accessed 25-March-2018].

[18] B. Goetz, "Java theory and practice: Concurrent collections classes," 2003.

[19] S. Goldsmith and D. S. Wilkerson. Stage.tigris.org - open source software engineering tools. [Accessed 20-March-2018]. [Online]. Available: http://trend-prof.stage.tigris.org/

[20] S. Gulwani, K. K. Mehra, and T. Chilimbi, "Speed: precise and efficient static estimation of program computational complexity," in *ACM Sigplan Notices*, vol. 44, no. 1. ACM, 2009, pp. 127–139.

[21] S. Halim, Z. C. Koh, V. B. H. LOH, and F. Halim, "Learning algorithms with unified and interactive web-based visualization." *Olympiads in Informatics*, vol. 6, 2012.

[22] L. Hickman, "How algorithms rule the world," *The Guardian*, vol. 1, 2013.

[23] Java-Oracle. Class runtime. [Accessed 7-April-2018]. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html

[24] JEdit. Jedit - programmer's text editor. [Accessed 3-April-2018]. [Online]. Available: http://www.jedit.org/index.php?page=features

[25] K. Kalorkoti, "Inf2b: Priority queues and heaps," [Accessed 7-April-2018]. [Online]. Available: https://www.inf.ed.ac.uk/teaching/courses/inf2b/algnotes/note06.pdf

[26] ——, "Inf2b: Sorting, merge sort and the divide-and-conquer technique," [Accessed 7-April-2018]. [Online]. Available: https://www.inf.ed.ac.uk/teaching/courses/inf2b/algnotes/note07.pdf

[27] M. Kay, *XSLT 2.0 and XPath 2.0 Programmer's Reference*. John Wiley & Sons, 2011.

[28] Knowm. Package org.knowm.xchart. [Accessed 7-April-2018]. [Online]. Available: https://knowm.org/javadocs/xchart/index.html

[29] ——. Xchart – simple java charts. [Accessed 7-April-2018]. [Online]. Available: https://knowm.org/open-source/xchart/

[30] ——. Xchart repository. [Accessed 7-April-2018]. [Online]. Available: https://github.com/knowm/XChart

[31] D. Knuth, "The art of computer programming 1: Fundamental algorithms 2: Seminumerical algorithms 3: Sorting and searching," *MA: Addison-Wesley*, vol. 30, 1968.

[32] D. E. Knuth, "Sorting and searching, 2nd edn. the art of computer programming, vol. 3," 1998.

[33] LibreOffice. Libreoffice - the document foundation. [Online]. Available: https://www.libreoffice.org/

[34] M. Loy, R. Eckstein, D. Wood, J. Elliott, and B. Cole, *Java swing*. " O'Reilly Media, Inc.", 2002.

[35] S. J. Metsker, *The Design Patterns Java Workbook*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[36] B. N. Miller and D. L. Ranum, *Problem Solving with Algorithms and Data Structures Using Python SECOND EDITION*. Franklin, Beedle & Associates Inc., 2011.

[37] Y. W. Nevena Blagoeva, Eziama Ubachukwu and S. Rachapudi, "Latest github repository of algobench." [Online]. Available: https://github.com/nevenaBlagoeva/AlgoBench

[38] Oracle. Java api – class concurrentmodificationexception. [Accessed 7-April-2018]. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/util/ConcurrentModificationException.html

[39] J. A. Oracle. Algorithm and data structure visualisation (adv). [Accessed 3-April-2018]. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/javax/swing/JTextPane.html

[40] L. Prechelt, "An empirical comparison of seven programming languages," *Computer*, vol. 33, no. 10, pp. 23–29, 2000.

[41] S. Rachapudi, "Algorithms workbench for inf2b." Master's thesis, School of Informatics, University of Edinburgh, 2017.

[42] C. Segura, I. Pita, R. del Vado Vírseda, A. I. Saiz, and P. Soler, "Interactive learning of data structures and algorithmic schemes," in *International Conference on Computational Science*. Springer, 2008, pp. 800–809.

[43] A. Shukla, S. Chaturvedi, and Y. Simmhan, "Riotbench: a real-time iot benchmark for distributed stream processing platforms," *arXiv preprint arXiv:1701.08530*, 2017.

[44] S. S. Skiena, *The algorithm design manual (2nd ed.)*. Springer Science & Business Media, 2010.

[45] B. Stroustrup, *Programming: principles and practice using C++*. Pearson Education, 2014.

[46] S. S. J. G. top contributors Arka Prava Basu, Kevin Nadro. Algorithm visualizer. [Accessed 20-March-2018]. [Online]. Available: http://algo-visualizer.jasonpark.me/#path=backtracking/knight's_tour/basic

[47] E. Ubachukwu, "Algorithms workbench for inf2b." Master's thesis, School of Informatics, University of Edinburgh, 2015.

[48] H. Van Vliet, H. Van Vliet, and J. Van Vliet, *Software engineering: principles and practice*. Wiley New York, 1993, vol. 3.

[49] Y. Wang, "Algorithms workbench for inf2b." Master's thesis, School of Informatics, University of Edinburgh, 2016.

[50] Xeiam. Xchart (xeiam) – package summary. [Accessed 7-April-2018]. [Online]. Available: http://xeiam.com/javadocs/xchart/index.html?com/xeiam/xchart/package-summary.html