QuizGen

A project presented as part of the coursework for CSC3003S in the Department of Computer Science

University of Cape Town

Sabir Buxsoo
Department of Computer Science
bxsmuh001@myuct.ac.za

Sicelokuhle Shabalala
Department of Computer Science
shbsic001@myuct.ac.za

Mlungiseleli Notshokovu
Department of Computer Science
ntsmlu003@myuct.ac.za

Abstract

This report outlines looks at the implementation of the Quiz Generator program for the CSC3003S course. The report starts by describing the project and then goes into the functional and non-functional requirements. Then we look at the use cases and describe the features of the solution in detail. The report then outlines the design and the architecture of the program followed by a through breakdown of the implementation process. Finally the testing and verification is outlined, followed by some final thoughts on the project.

1. Introduction

The Vula Learning Management System, powered by Sakai, is used at the University of Cape Town. It has a Tests and Quizzes feature which is a tool which allows lecturers to test students by providing them with different, but similar, questions such as Multiple Choice or True or False questions. However, the process of creating those quizzes is time consuming.

As such, the goal of this project is to create a software which will allow lecturers to generate random questions on Data Structures which are covered in the CSC2001F lectures, in an attempt to prevent plagiarism among peers by having different variations of the same question for each of the students in a class. These generated questions will test the students' abilities to insert and delete items into various data structures covered in the CSC2001F course.

The system provides a command line interface for the lecturer to interact with, to generate specified number of questions on Data Structures such as Priority Queues, Hash Tables, Binary Search Trees and AVL Trees. The software then generates the questions into a formatted text file as required by the Vula Tests and Quizzes tool.

We approached the project by first gathering the user requirements from our client and gaining a good understanding of the problem to solve and what kind of solution our client wanted to have. We adopted a Vertical Evolutionary Prototype during the early stage of the project as it allowed us to focus on the core features. The evolutionary prototype allowed us to demonstrate a working prototype to our client and get feedback to improve the solution to the client's vision. This provided us with the benefit of testing and refining our core features while allowing our client to be part of our development process. Throughout the course of the development process, we made use of Agile Methodologies and Test Driven Development to ensure that each of the features we implemented were working properly before moving on to the next.

Given the nature of the project, which required the implementation of several common Data Structures, research was done into several Data Structures to gain an understanding of how to implement them into our project. The resources we used were mostly the website Geeksforgeeks.com [1] to learn implementation of the common Data Structures, the book Data Structures and Algorithm Analysis in Java by Mark Allen Weiss [2] and the book Algorithms by C.L.R.S [3]. Those resources helped us not only implement the common Data Structures, but also gave us insight on how to design questions from scratch to test the understanding of students who will be answering those questions on Vula.

2. Requirements Captured

A thorough analysis was conducted by the team to gather the functional, non-functional and usability requirements for this project. Those requirements ensured that the final deliverable met those requirements.

2.1. Functional Requirements

Table 1 below summarises the Functional Requirements.

Table 1: *Summary of Functional Requirements*

Functional Requirement	Description
1. Data Structure Choice	The system should be able to take the Data Structure choice input from the user to show the questions for the selected Data Structure
2. Question Choice	The system should be able to take the question choice input from the user
3. Number of random questions	The system should be able to take the input of number of questions the user wants generated
4. Filename	The system should be able to take the input specified by the user for the filename which will be output and contain questions
5. Generate Questions	The system should be able to generate random questions for the following Data Structures: a. Binary Search Tree: Insertion, Deletion, and General Questions b. AVL Tree: Insertion, Deletion, and General Questions c. Hash Tables: Insertion, Deletion and General Questions d. Priority Queues: Insertion, Deletion
6. Write to output text file and store	The system should be able to write questions into a text file and store the file on the user's computer in a specified directory called `generatedQuestions`
7. Input to go back or exit	The system should be able to take a user input to take the use back to a previous menu or exit the program.

2.2. Non-Functional Requirements

Table 2 below summarises the non-functional requirements.

Table 2: *Summary of Non-Functional Requirements*

Non-Functional Requirement	Description
1. Reliability	The system should be reliable and provide the correct set of questions with the correct answers
2. Usability	The system should have a simple flow to allow the user to do the tasks efficiently
3. Portability	The system should be portable, so that it can be used on different operating systems without any issues.
4. Performance	Questions should not take too long to be generated and output into the text file
5. Stability	The system should be stable as it handles deals with generating questions which are used for tests
6. Maintainability	The system should be easily maintainable in case more questions need to be added in the future or bugs need to be fixed
7. Reusability	The system should make use of modular, re-usable code

2.3. Use Cases

The only Actor will be the User (Lecturer) who will be interacting with the application. Questions are pre-defined in the program. To avoid confusion, User and Lecturer are used interchangeably throughout and hold the same meaning.

2.3.1. **Actor:** User/Lecturer

Action: Launch the program and choose option from list of Data Structures
The user launches the program and is presented with a screen which displays a series of Data Structures for the user to choose from: Binary Search Tree, AVL Tree, Hash Tables, Priority Queue. The user can choose the option by inputting the option value in the command line.

Typical Course of Events:
The program launches successfully and presents the list of Data Structures show up. Selecting the main category option, opens the sub-category options.

Alternative Course of Events:
The program fails to launch or the program crashes on launch.

2.3.2. **Actor:** User/Lecturer
Action: Choose a specific Data Structure
The user makes a choice from the main menu for a specific Data Structure. The lecturer is the presented with a Question menu about that specific data structure (e.g Binary Search Tree), which contains a list of sample questions for the user to choose from. A Back option from user input is available to go to the previous Menu.

The lecturer selects a question from the list. They can only select one question at once to be generated.

Typical Course of Events:
The Question menu is opened successfully without interruptions, crashes and errors, and all the options for said menu(mentioned above) are displayed successfully. Back works as expected. Question option is successfully input.

Alternative Course of Events:
The Question menu fails to open and the system crashes, showing and error to the user. The program exits unexpectedly. Questions list are not displayed. Back option does not work.

2.3.3. **Actor:** User/Lecturer
Action: Select question to be generated
The user makes a choice of the question they want generated from the list in the prompt, by inputting their choice number in the terminal.

Typical Course of Events:
The user is prompted to input the number of variations of the question they want generated.

Alternative Course of Events:
The input prompt does not work and input values cannot be entered.

2.3.4. **Actor:** User/Lecturer
Action: Input the number of variations of questions to be generated
The use inputs the number of random variations of that specific question they want to generate. They can enter this number in the prompt.

Typical Course of Events:
The user successfully enters the number of questions to be generated. The user is now prompted to enter the fine name to which they will want to save their generated questions.

Alternative Course of Events:
The input prompt does not work and input values cannot be input.

2.3.5. **Actor:** User/Lecturer
Action: Set file name for output

The lecturer can enter the Output File Name for the file to be generated in the prompt.

Typical Course of Events:
The user successfully enters the filename of the output file to be generated and the user sees a message which shows the location in which the file will be saved.

Alternative Course of Events:
The input prompt does not work and filename cannot be entered.

2.3.6. **Actor:** User/Lecturer
Action: Generate questions
After selecting a specific question, inputting the number of variations of the question they want generated, and setting up the filename, the lecturer presses Enter and they are presented with a final prompt to ask them to confirm whether they want to proceed or not using y/n for yes/no respectively.

Typical Course of Events:
Once the user inputs y, a file is generated with the list of questions according to the lecturer's specifications and which follows the guidelines for submission on the Vula Tests and Quizzes tabs. The lecturer is presented with a success message and the location where the file is saved.

If the Lecturer inputs n, the action is cancelled and the lecturer can go back to make changes or close the program.

Alternative Course of Events:
The generating of the output fails to work. On inputting y the file is not generated which renders the program useless. On inputting n, unexpected crashes or errors occur.

2.3.7. **Actor:** User/Lecturer
Action: Inspect Output File
The lecturer can see the generated output file with the questions to ensure questions have been successfully generated.

Note: This is an action which typically happens outside the program/software. However, it was included here for completeness.

2.3.8. **Actor:** User/Lecturer
Action: Close Program
The user must be able to exit and close the program at any point during a normal working flow and at any step, with the exception when a file is currently being generated.

Typical Course of Events:
When user the program exits gracefully without errors.

Alternative Course of Events:
The Close button does not work. Program exits with errors.

3. Design Overview

During the early stage of the project, we built a naïve prototype of a solution based on the requirements gathered from the client. This allowed us to demo the prototype to the client to gain feedback on whether we were headed in the right direction and also allow us to gain a better view as to how the final solution would look like. After getting an overall view of the project, we designed our final solution based on the following features:

- Command-line interface (CLI) to interact with the program
- Inputs are made via the CLI
- User will have a list of Data Structures to choose from
- User can make choice of Data Structure
- User can select the question they want to generate for a specific Data Structure
- User can specify the number of questions they want generated
- User can specify the file name in which the generated questions will be saved

Figure 1 below shows the design class diagram of the final solution. We ensured that the code was built to be modular and re-usable.

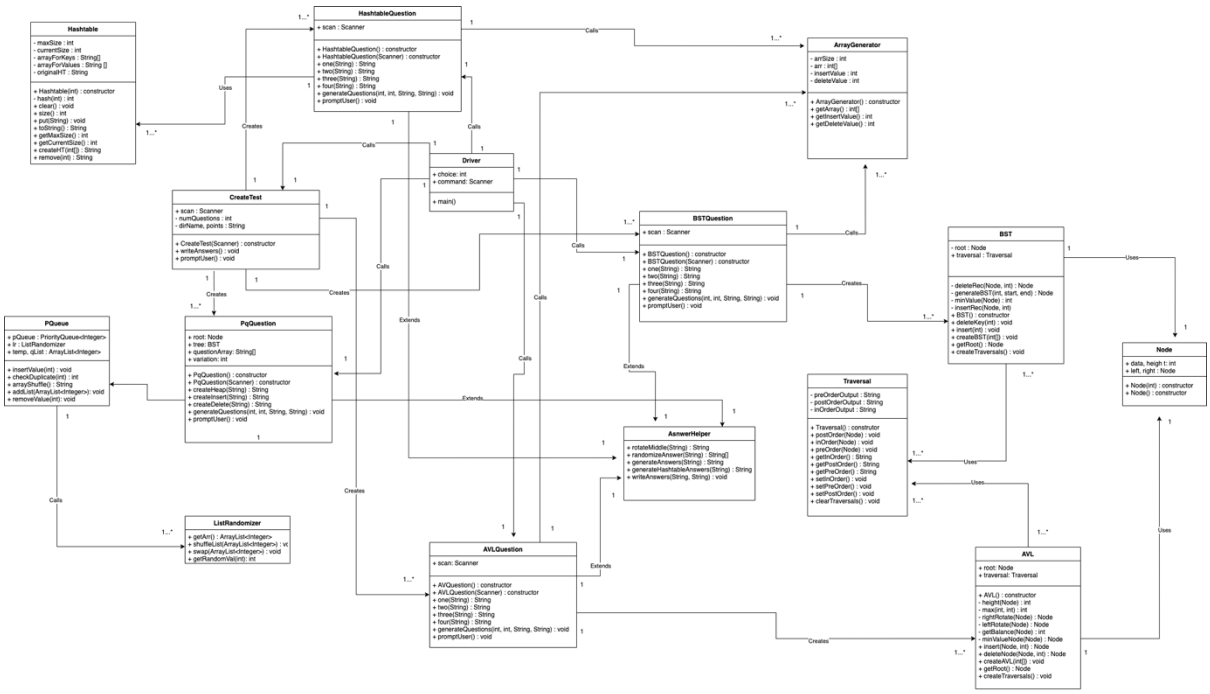


Figure 1: Design Class diagram of final solution

3.1. Algorithms and Data Organisation

Given the nature of this project, no external database was used to store the question to be generated for each Data Structure. We made the strategic decision, based on our client requirements, to not use an external database to store the questions to avoid complexity in the program and also due to the limited number of questions.

As such, we decided to instead store the questions directly in the program source code, but we made sure to consider separation of concerns and thus created different classes which would store the question for each data structure. This design approach allowed us to keep the code clean and modular, and ensure that we can debug easily. It also makes it possible to add questions to the code base in the future. The different classes will be discussed in further details the Implementation Section below.

As far as algorithms are concerned, each Data Structure was implemented using the approach from trusted sources [1][2][3]. This ensured the implementation for each Data Structure was efficient and worked as expected.

Figure 2 below shows a high-level architecture diagram of our system from a design point of view. We chose a layered approach, and although we do not have a database management system, we included a package to separate our question classes to give the reader a high-level outline of what the architecture looks like. It must be noted that since no database was used, the architecture design will not be as a typical design. We represent the Questions we stored separately in Figure 2 to demonstrate the separation of concerns.

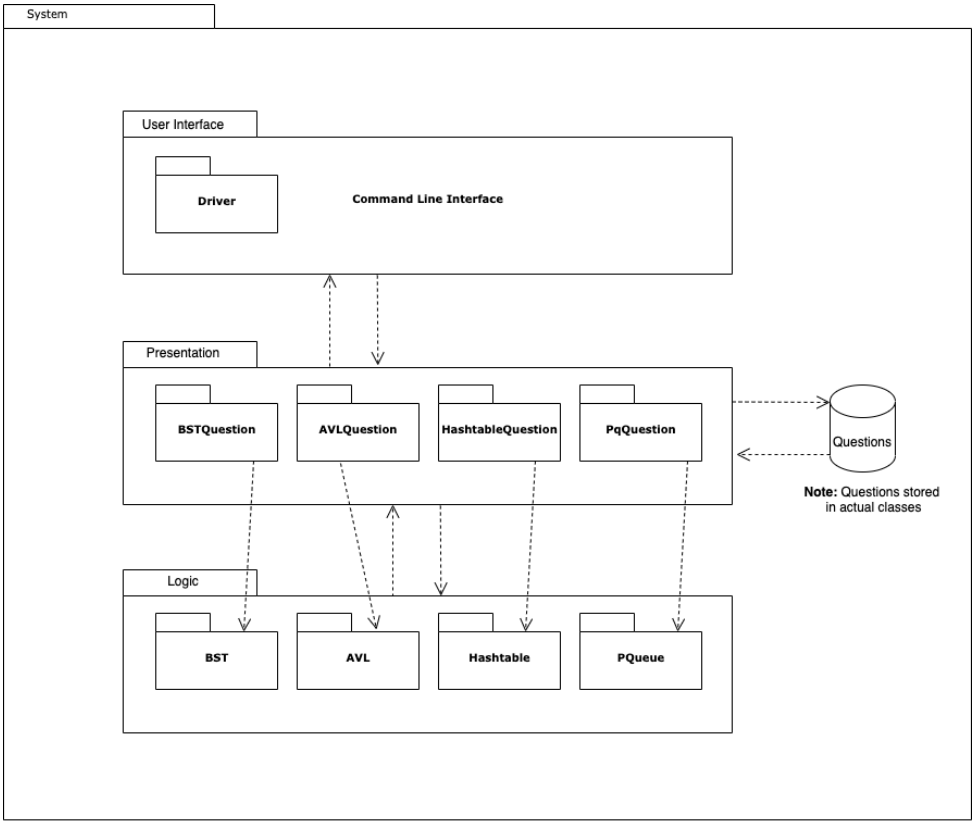


Figure 2: High-level architecture diagram of the system

4. Implementation

The project was intended to be used by lecturers of the CSC2001F course at the University of Cape Town, to generate questions for the course. As such, there was no heavy focus on User-Interface design as our client made it known that they are comfortable with using the Command-Line Interface.

The solution was implemented using the Java Version 8 programming language. We found out that the language was suited to build the solution as it is a class-based object-oriented programming language. Java Version 8 also provides a lot of features and inbuilt methods which helped building the solution much more efficiently and in a modular fashion.

Given the nature of this project, no Database solution was used for persisting data as we had a limited number of questions which needed to be constructed using different algorithms and functions to be generated in the first place. As such, the questions were instead written in their own separate classes for each Data Structure and each question is created as and when required, as explain in Section 3.1.

Since the project revolves around generating questions for the common Data Structures, our solution relied heavily on referencing several Data Structures and their implementations from reliable sources. In this section, we will look at the different Data Structures used throughout

the project. We will also look at some important classes which were necessary to ensure the solution worked as expected.

The State Diagram in Figure 3 below shows the overall flow of the application and the give the user an a description of the behaviour of the system.

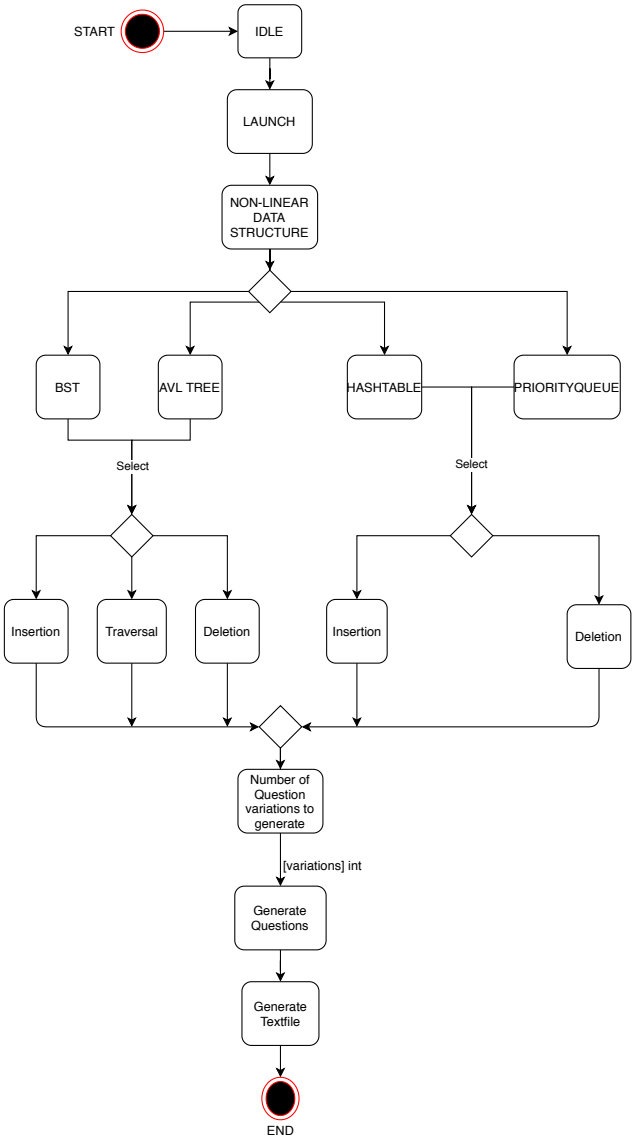


Figure 3: State diagram to depict flow of system

4.1. Binary Search Tree Data Structure

An important Data Structure which is taught to students is the Binary Search Tree(BST). For the purpose of this project, the BST was implemented as a standalone class, called BST.java. The class contained important methods to create a Balanced BST from a sorted array [4]. This was done on purpose to avoid generating unbalanced BSTs and have a situation where all the nodes are to the left or right.

The Binary Search Tree was implemented to generate questions on the topic of BST which will be used to test student's knowledge of Traversals, Insertion and Deletion in a BST.

For the Binary Search Tree Data Structure, several classes were created and these are outlined in Table 3 below.

Table 3: *Summary of Classes for BST*

Class name	Purpose
1. BST	This is the class which contains the methods to create a BST from a sorted array. It also contains methods for the insertion and deletion of elements from the tree. This class also makes use of the Node.java class as well as the Traversal.java class. Those are defined in section 3.6 below. Some important methods in this class are the generateBST() which creates a fresh BST. There is also the deleteRec() and insertRec() which handle recursive deletion and insertion from and into the BST respectively.
2. BSTQuestion	This class was designed to contain the different questions involving the Binary Search Tree. This was made as a separate class to keep the code modular and ensure easy bug fixes and addition of questions in the future. This class makes use of the BST.java class as well as other helper classes defined in section 3.6 below.

4.2. AVL Tree Data Structure

The AVL tree is defined as a self-balanced Binary Search Tree where the difference in heights of the left and right subtrees cannot be greater than one for all the nodes . In a similar fashion to the BST, the AVL Tree Data Structure was implemented to generate questions on the topic of AVL which will be used to test student's knowledge of whether they can create a tree, their knowledge of tree traversals, insertion and deletion in an AVL tree.

The AVL tree as implemented in the class called AVL.java which contains all the methods to create an AVL tree, do insertions and deletions.

For the AVL tree, several classes were created and they are outlined in Table 4 below.

Table 4: *Summary of Classes for AVL*

Class name	Purpose
1. AVL	This is the class which contains the methods to create a AVL tree. It also contains methods for the insertion and deletion of elements from the tree. This class also makes use of the Node.java class as well as the Traversal.java class. Those are defined in section 3.6 below. Some important methods for this class are the createAVL() which makes use of the insert() method to create a fresh AVL tree. The insert() method is used to insert future elements and rebalance the tree. The deleteNode() method is also important as it handles deletion of a node from the tree.
2. AVLQuestion	This class was designed to contain the different questions involving the AVL Tree. This was made as a separate class to keep the code modular and ensure easy bug fixes and addition of questions in the future. This class makes use of the AVL.java class as well as other helper classes defined in section 3.6 below.

4.3. Hashtable Data Structure

The Hashtable is a very interesting data structure which is concerned with the implementation of a structure that maps keys and values. For the purpose of this project we implemented a version of Hash table and took the approach of handling collisions using the Linear Probing method. We ensured the questions were designed to be done within a short time by the students. As such, we considered insertion and deletion of Hashtables sizes 7 and 10 respectively and always input 7 elements into the Hashtable and delete 1.

For deletion, it is specially important to mention that we considered only the Lazy Deletion option which basically replaces the deleted value with a "-" to demonstrate the slot is now empty.

For the Hashtable, several classes were created and they are outlined in Table 5 below.

Table 5: *Summary of Classes for Hashtable*

Class name	Purpose
1. Hashtable	This is the class which contains the methods to create a Hashtable. It also contains methods for the insertion and deletion of keys in and out of the Hashtable. Some important methods for this class are the createHT() method which handles the creation of a Hashtable with Linear Probing insert. The remove() method is also important as it is concerned with the removal of a key from the Hashtable and replacing it with a hyphen to demonstrate Lazy Deletion.
2. HashtableQuestion	This class was designed to contain the different questions involving the Hashtable. This was made as a separate class to keep the code modular and ensure easy bug fixes and addition of questions in the future. This class makes use of the Hashtable.java class as well as other helper classes defined in section 3.6 below.

4.4. Priority Queue Data Structure

For this project we implement the PriorityQueue with Min-heap to test the student's knowledge of insertion, deletion and whether they can construct a Priority Queue from scratch. We make use of the Java 8 programming language in-built PriorityQueue feature to implement this Data Structure.

Table 6: *Summary of Classes for AVL*

Class name	Purpose
1. PQueue	This is the class which contains the methods to create a PQueue. It also contains methods for the insertion and deletion of elements in the Priority Queue. Some important methods for this class are addList() which creates a Priorirty queue. Then ther insertValue() and deleteValue() methods are used for insert and delete from the Priority queue respectively.
2. PqQuestion	This class was designed to contain the different questions involving the Priority Queue. This was made as a separate class to keep the code modular and ensure easy bug fixes and addition of questions in the future. This class makes use of the PQueue.java class as well as other helper classes defined in section 3.6 below.

4.5. Bonus Feature – Create a test

Since the project is designed for our client to generate pools of questions for tests, we thought it would be a good idea to give the user an option to generate a full test package. What this basically does it allows the client to specify the number of questions they want to generate for each data structure, they input the number of variations for each question and enter a folder name in which all the generated files will be saved. This then generates a pool of questions for all the Data Structures and places them in the newly created folder.

For example, for an upcoming test, the lecturer can generate a `Test01` folder which contains the pol of questions for each data structure. This allows for a fully randomized test to be created for multiple student within a matter of seconds.

Table shows the class for the bonus feature.

Table 7: *Summary of helper classes and other additional classes*

Class name	Purpose
CreateTest	This class contains the important method promptUser() which basically prompts the user to enter their preferences and then it creates all the questions for the different data structures and saves them. This code is particularly long due to it being a full generator.

This feature is shown in the User Manual (Annex A).

4.6. Other classes

In addition to the classes described above, several other classes were created to be used by other classes. Some of those classes were necessary for the correct working of another class (e.g. Node.java). Other classes were used as helper classes. Those classes are described in Table 7 below.

Table 8: *Summary of helper classes and other additional classes*

Class name	Purpose
1. AnswerHelper	This class is used by the PqQuestion, HashtableQuestion, BSTQuestion, the AVLQuestion. It basically is a simpler helper class which generates randomized versions of the original answer to make sure there are different answer options for the multiple choice questions which are generated.
2. Node	This class is used by the BST.java and the AVL.java classes. Those classes require the Node.java to function. This is a very important class as the BST and AVL tree require this class to perform their tasks.
3. ArrayGenerator	This class generates a random array as well as two values. This class is used to help the AVL Hashtable and BST classes as it provides an array of integers which are used to create the AVL and BST trees. This class also provides a randomly generated value which will be inserted in a created tree. It also provides a random value from the generate array which will be used by the AVL and BST to remove the element from a tree.
4. Traversal	The Traversal class is used the the AVL and BST classes to provide the InOrder, PreOrder and PostOrder of a tree. This is a specially important class as it is used heavily to create questions.
5. Driver	This is the main class of the program. This is where the program starts its execution and takes commands from the user. This class prompts the user to select the Data Structure for which they want to generate questions.
6. ListRandomizer	This class behaves like the ArrayGenerator class but is separate as it is only used by the PQueue class which requires ArrayLists instead of Arrays.

5. Program Validation and Verification

The key to a successful project is to ensure that the final product is thoroughly tested before delivery to the client. This ensures that the solution behaves as expected, is robust and does not contain logical errors.

Given the nature of the project was to implement Data Structures, it was important to test that the output of the implemented Data Structures were correct. In this section, we describe the testing approaches we used and give an outline of the test cases.

5.1. Summary of testing cases

The table below shows the Summary of our testing plan.

Table 9: *Summary of testing plan*

Process	Approach
1. Class testing	Each class was tested individually to ensure the methods were working as expected. This was done using the White-box testing approach by looking at the individual methods and checking they correctness. If a method was not giving the correct behaviour, the test failed and would require fixes. This was done manually and by making us of trace debugging techniques (using System.out.println, for example).
2. Integration testing	Here as well we made use of the White-box testing technique to test the coupling and cohesion of our classes. Since the code is very modular, it is very important for classes to behave as expected and work perfectly with other classes. We conducted this test manually as well by using tracing debugging techniques. We also tested the program as a whole to see if the classes worked together. Here we used a black-box testing technique to see if the program achieved expected behavior.
3. Validation testing	For this testing phase, we used the black-box testing technique, which is basically a form of acceptance testing, to test whether our client's requirements were met. This

	meant running the program multiple times and going through the various flows of the program to ensure all features and functionalities were behaving as expected.
4. System testing	This testing was done to check the behaviour of the program to ensure it was robust and portable. This was done by running the program on various machines to test their portability across different operating systems using Java Version 8. We also tested the robustness of the program by inputting large number of questions to be generated to find whether or not the program would crash when generating large sets of questions (e.g 100,000<=n <=1,000,000) where is is the number of question variation to be generated.

5.2. Summary of tests carried out

Test Description	**Data Set**	**Reason of Choice**	**Test Cases**		
			Normal Function	**Extreme Boundary Case**	**Invalid Data**
Input choice of Data Structure	1,2,3,4,5,0 (input commands)	To check if user can input commands in the terminal	User is redirected to next step of the program flow with data 1,2,3,4,5	With 0, program exits	The program asks the user what they want to do again
Binary Search tree select options by input	1,2,3,4(input commands)	To check if user can input commands in the terminal to select their choice of question	User prompted with further steps to generate questions	n/a	The program exits with error
Hashtable select options by input	1,2,3,4(input commands)	To check if user can input commands in the	User prompted with further steps to	n/a	The program exits with error

		terminal to select their choice of question	generate questions		
AVL tree select options by input	1,2,3,4(input commands)	To check if user can input commands in the terminal to select their choice of question	User prompted with further steps to generate questions	n/a	The program exits with error
Priority Queue select options by input	1,2,3 (input commands)	To check if user can input commands in the terminal to select their choice of question	User prompted with further steps to generate questions	n/a	The program exits with error
Test if output files contain questions	Data is generated from the program and are questions which are predefined in the BSTQuestion, AVLQuestion, HashtableQuestion and PqQuestion classses	To check output files contain the questions as expected	Output files contain generated questions	n/a	Output files are empty, Output files are not generated
Check if output file generated	n/a	To check if output file is created in the correct directory	Directory contains output files	n/a	Directory does not contain output files
Binary Search tree validity (See Appendix B)	Binary search tree node, traversals	To check if the tree provided is actually a BST	Tree is valid	n/a	Tree invalid
AVL tree validity (See Appendix B)	AVL tree node, traversals	To check if the tree provided is actually an AVL	Tree is valid	n/a	Tree invalid

Hashtable validity (See Appendix B)	Array of data, size of hashtable	To check if hashtable is valid after insert and delete	Hashtable is valid	n/a	Hashtable invalid
Priority queue validity (See Appendix B)	ArrayList	To check if the priority queue is valid	Priority queue is valid	n/a	Priority queue is invalid

6. Conclusion

After a thorough analysis and testing of the final product, we concluded that the solution was behaving as expect, robust and portable. The objective was to create a program which would allow the generation of questions involving Data Structures and we achieved the result to completion and this was heavily influenced by adoption an initial vertical evolutionary prototype approach right at the beginning to understand the full client requirements.

We also understood the important of writing modular code and ensuring that the code is maintainable. The modular code allowed us to devise a feature which will potentially allow lecturers to generate Tests Question Pools within matter of seconds.
We also made use of Test-Driven development to ensure the features were working correctly and the code was refactored to ensure consistency across the board.

Unfortunately, we realised that Vula is very limited in terms of what can be uploaded as question and as such we encountered some limitations which made it difficult to generate more complex questions, especially on graphs data structures which require images. More complex tree and hashtable questions could also not be generated.

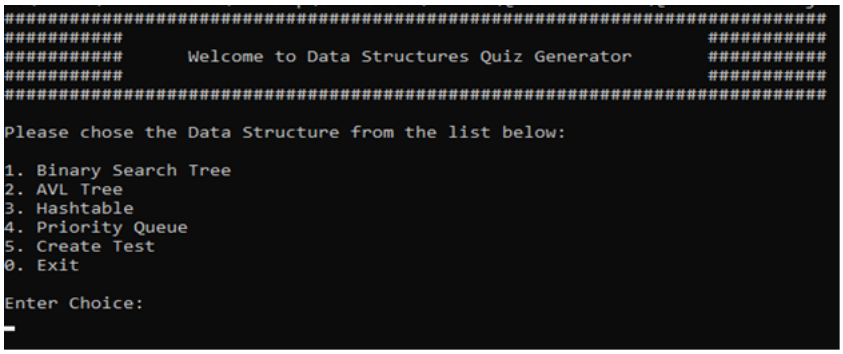
In conclusion, the project was deemed successful although communication could have been better among team members. We had a steep learning curve and managed to apply the concepts of Software engineering to build a fully functional product.

Annex A – User Manual

1. Compile the code in terminal/command using “javac” as shown below. Please ensure the generatedQuestions directory is never deleted. Another option is to run `make` in the directory if you are on a Unix machine.

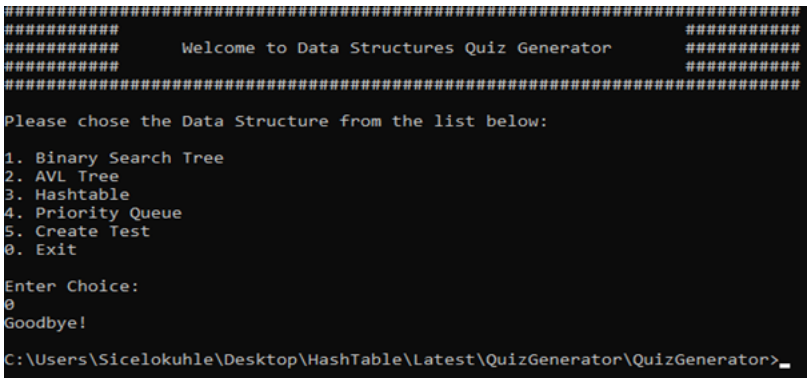
2. Run the code using “java” in terminal/command as shown below.

3. After performing the above two operations you’ll be redirected to the following screen:



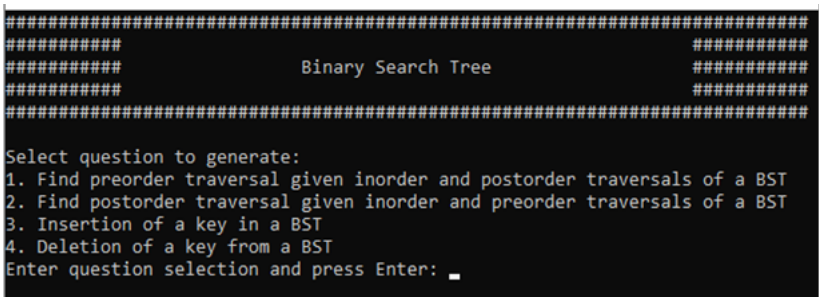
4. Here you’re prompted to select a data structure for which you want to generate a question for. There are 4 Non-linear data structures (Binary Search Tree, AVL Tree, Hashtable and Priority Queue) to choose from. Select a number (1,2,3,4,5 or 0) when you press 0 the program terminates.

5. Selecting “0. Exit”: The program must terminate.



Note: To run the program again for a different option to choose, repeat step 1 and 2.

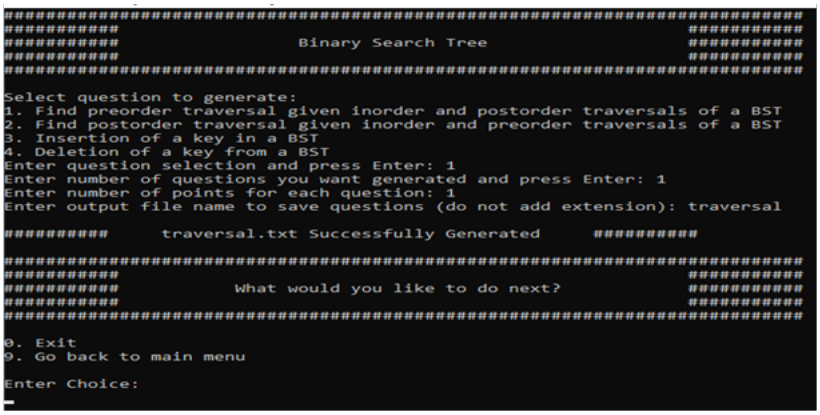
Selecting “1. Binary Search Tree”: The program should redirect you to the following screen.



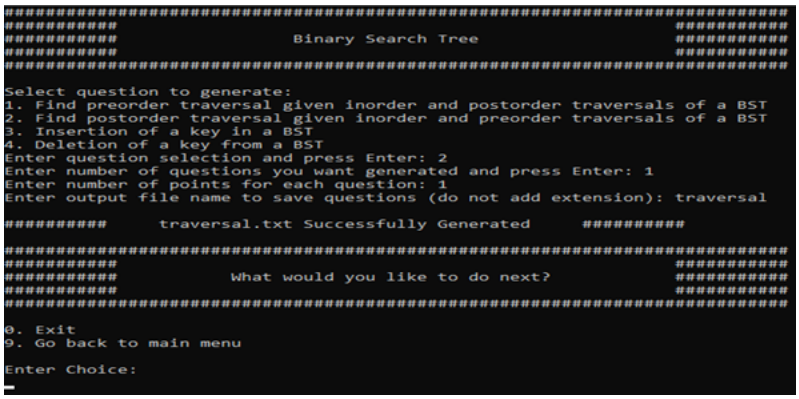
6. Here you are presented with types of question to generate i.e. Insertion, Deletion or Traversal.

7. Select 1 and 2 from above.

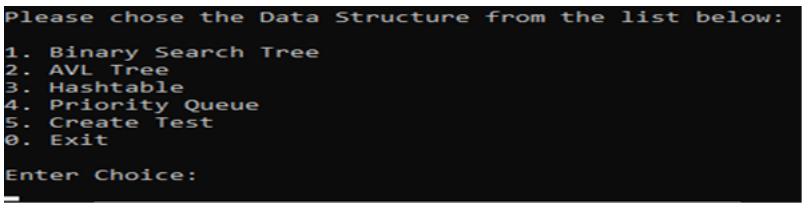
8. After Selecting 1. You’ll prompted to Exit or Go back to main menu, choose any.



9. After Selecting 2. You'll be prompted to Exit or Go back to main menu, choose any.



10. Selecting 9 redirects you to the main menu.



The tables below give 1 sample question for each operation available to each data structure and the expected outcome (Correct Answer), this is done for the user to perform manual analyses of the performance of the data structure.

Table 1: *Binary Search Tree Sample Questions*

Binary Search Tree (Operation)	**Preview Question**	**Answers to Choose from**	**Correct Answer**
Pre-Order Traversal	Suppose you have a Binary Search Tree with In-Order Traversal 4 68 75 82 86 88 99 and Post-Order Traversal 4 75 68 86 99 88 82. What is the Pre-Order Traversal of the Tree?	A. 82 68 4 75 88 86 99 B. 82 4 75 88 86 68 99 C. 82 75 88 86 68 4 99 D. 82 88 86 68 4 75 99 E. None of the answers are correct	A. 82 68 4 75 88 86 99
Post-Order Traversal	Suppose you have a Binary Search Tree with inorder traversal 25 29 74 79 82 92 94 and preorder traversal 79 29 25 74 92 82 94. What is the postorder traversal of the tree?	A. 25 74 29 82 94 92 79 B. 25 29 82 94 92 74 79 C. 25 82 94 92 74 29 79 D. 25 94 92 74 29 82 79 E. None of the answers are correct	A. 25 74 29 82 94 92 79
Insertion	Suppose you have a Binary Search Tree with inorder traversal 15 27 32 64 73 75 86 and postorder traversal 15 32 27 73 86 75 64. Now suppose you insert 71 in the tree. What is the preorder traversal of the resulting tree after the insert?	A. 64 27 15 32 75 73 71 86 B. 64 15 32 75 73 71 27 86 C. 64 32 75 73 71 27 15 86 D. 64 75 73 71 27 15 32 86 E. None of the answers are correct	A. 64 27 15 32 75 73 71 86
Deletion	Suppose you have a Binary Search Tree with inorder traversal 4 21 44 51 76 86 93 and postorder Traversal 4 44 21 76 93 86 51. Now suppose you Delete 76 from the tree. What is the preorder Traversal of the resulting tree after the deletion?	A. 51 21 4 44 86 93 B. 51 4 44 86 21 93 C. 51 44 86 21 4 93 D. 51 86 21 4 44 93	A. 51 21 4 44 86 93

Table 2: *AVL Tree Sample Questions*

AVL Tree (Operations)	**Preview Question**	**Answer to Choose from**	**Correct Answer**
Pre-Order Traversal	Suppose you have an AVL Tree with inorder traversal 8 29 32 48 61 66 76 and postorder traversal 8 32 29 61 76 66 48. What is the preorder traversal of the tree?	A. 48 29 8 32 66 61 76 B. 48 8 32 66 61 29 76 C. 48 32 66 61 29 8 76 D. 48 66 61 29 8 32 76 E. None of the answers are correct	A. 48 29 8 32 66 61 76
Post-Order Traversal	Suppose you have an AVL Tree with inorder traversal 21 23 26 53 54 80 88 and preorder traversal 53 23 21 26 80 54 88. What is the postorder traversal of the tree?	A. 21 26 23 54 88 80 53 B. 21 23 54 88 80 26 53 C. 21 54 88 80 26 23 53 D. 21 88 80 26 23 54 53 E. None of the answers are correct	A. 21 26 23 54 88 80 53
Insertion	Suppose you have an AVL Tree with Inorder traversal 2 19 29 32 41 94 95 and Postorder traversal 19 2 32 29 94 95 41. Now suppose you insert 11 in the tree. What is the Preorder Traversal of the resulting tree after the insert?	A. 41 29 11 2 19 32 95 94 B. 41 11 2 19 32 95 29 94 C. 41 2 19 32 95 29 11 94 D. 41 19 32 95 29 11 2 94 E. None of the answers are correct	A. 41 29 11 2 19 32 95 94
Deletion	Suppose you have an AVL Tree with Inorder traversal 1 44 59 71 72 93 99 and Postorder traversal 1 44 71 93 99 72 59. Now suppose you Delete 72 from the tree. What is the Preorder Traversal of the resulting tree after the deletion?	A. 59 44 1 93 71 99 B. 59 1 93 71 44 99 C. 59 93 71 44 1 99 D. 59 71 44 1 93 99 E. None of the answers are correct	A. 59 44 1 93 71 99

Table 3: *Hashtable Sample Questions*

Hashtable (Operations)	**Preview Question**	**Answer to Choose from**	**Correct Answer**
Insertion via Linear Probing using Hash function h(x) = x%7	What is the resulting hashtable after inserting the keys [9, 44, 36, 62, 99, 45, 21] into an initially empty hashtable of length 7 using open addressing with hash function is h(x) = x % 7 and linear probing ?	A. Hashtable 1: 21 36 9 44 99 45 62 B. Hashtable 2: 21 9 44 99 45 36 62 C. Hashtable 3: 21 44 99 45 36 9 62 D. Hashtable 4: 21 99 45 36 9 44 62 E. None of the answers are correct	A. Hashtable 1: 21 36 9 44 99 45 62
Insertion via Linear Probing using Hash function h(x) = x%10	What is the resulting hashtable after inserting the keys [14, 7, 30, 41, 46, 73, 25] into an initially empty hashtable of length 10 using open addressing with hash function is h(x) = x % 10 and linear probing ?	A. Hashtable 1: 30 41 - 73 14 25 46 7 - - B. Hashtable 2: 30 - 73 14 25 46 7 - 41 - C. Hashtable 3: 30 73 14 25 46 7 - 41 - - D. Hashtable 4: 30 14 25 46 7 - 41 - 73 - E. None of the answers are correct	A. Hashtable 1: 30 41 - 73 14 25 46 7 - -
Lazy Deletion in hashtable of length 7	What is the resulting hashtable after deleting 31 from the hashtable 5 53 23 31 46 68 6 assuming LAZY DELETION?	A. Hashtable 1: 5 53 23 - 46 68 6 B. Hashtable 2: 5 23 - 46 68 53 6 C. Hashtable 3: 5 - 46 68 53 23 6 D. Hashtable 4: 5 46 68 53 23 - 6 E. None of the answers are correct	A. Hashtable 1: 5 53 23 - 46 68 6
Lazy Deletion in hashtable of length 10	What is the resulting hashtable after deleting 3 from the hashtable 17 51 - 43 3 - - 47 38 59 assuming LAZY DELETION?	A. Hashtable 1: 17 51 - 43 - - - 47 38 59 B. Hashtable 2: 17 - 43 - - - 47 38 51 59 C. Hashtable 3: 17 43 - - - 47 38 51 - 59 D. Hashtable 4: 17 - - - 47 38 51 - 43 59 E. None of the answers are correct	A. Hashtable 1: 17 51 - 43 - - - 47 38 59

Table 4: *Priority Queue Questions*

Priority Queue (Operations)	**Preview Question**	**Answer to Choose from**	**Correct Answer**
Insertion into empty priority queue	Min-Heap: What is the resulting Priority Queue from this list [19, 14, 24, 2, 30, 1, 7, 20, 21, 12]?	A. [1, 12, 2, 19, 14, 24, 7, 20, 21, 30] B. [1, 2, 19, 14, 24, 7, 20, 21, 12, 30] C. [1, 19, 14, 24, 7, 20, 21, 12, 2, 30] D. [1, 14, 24, 7, 20, 21, 12, 2, 19, 30] E. None of the answers are correct	A. [1, 12, 2, 19, 14, 24, 7, 20, 21, 30]
Insertion of a value into priority queue	Min-Heap: What is the resulting Priority Queue when the value 6 is inserted to the list [3, 12, 5, 13, 17, 16, 28, 24, 14]?	A. [3, 6, 5, 13, 12, 16, 28, 24, 14, 17] B. [3, 5, 13, 12, 16, 28, 24, 14, 6, 17] C. [3, 13, 12, 16, 28, 24, 14, 6, 5, 17] D. [3, 12, 16, 28, 24, 14, 6, 5, 13, 17] E. None of the answers are correct	A. [3, 6, 5, 13, 12, 16, 28, 24, 14, 17]
Deletion	Min-Heap: What is the resulting Priority Queue from deleting value 9 from the list [2, 7, 8, 10, 12, 13, 9, 30, 16]?	A. [2, 7, 8, 10, 12, 13, 16, 30] B. [2, 8, 10, 12, 13, 16, 7, 30] C. [2, 10, 12, 13, 16, 7, 8, 30] D. [2, 12, 13, 16, 7, 8, 10, 30] E. None of the answers are correct	A. [2, 7, 8, 10, 12, 13, 16, 30]

Annex B – Test Runs for Each Data Structure

Test Cases for Hashtable Linear Probing insertion:

1. Testing for Tablesize = 7
   Keys = [9,44,36,62,99,45,21]
   Modulo Operation:
   9%7 = 2
   44%7 = 2
   36%7 = 1
   62%7 = 6
   99%7 = 1
   45%7 = 3
   21%7 = 0

   Resulting Hashtable:
   0 21
   1 36
   2 9
   3 44
   4 99
   5 45
   6 62

2. Testing for Tablesize = 10
   Keys = [14,7,30,41,46,73,25]
   Modulo Operation:
   14%10 = 4
   7%10 = 7
   30%10 = 0
   41%10 = 1
   46%10 = 6
   73%10 = 3
   25%10 = 5

   Resulting Hashtable:
   0 30
   1 41
   2 -
   3 73
   4 14
   5 25
   6 46
   7 7
   8 -
   9 -

The hashtable works correctly since Resulting hashtables are similar to the Correct Answer computed manually as the above hashtable table. This proves that the algorithm is working. Deletion is a straight-forward operation, the value to be deleted is replaced by a flag (HYPHEN) to show it's been deleted. This can be seen on the program output directly.

Test Cases for Min-Heap Priority Queue:

Test case 1

Description: Check if given a randomly generated array list, it will return a newly ordered Priority Queue

Input: insert a randomly generated array [22, 4, 20, 25, 21, 15, 28, 10, 13]

Expected output: [4, 10, 15, 13, 22, 20, 28, 25, 21]

Test outcome: [4, 10, 15, 13, 22, 20, 28, 25, 21], Test passed

Test case 2

Description: Check if the multiple-choice questions do not have duplicates.

Input: insert a randomly generated array [6, 15, 28, 4, 24, 26] and checked if multiple questions are not duplicated.

Expected output: Four Distinct multiple-choice options

Test outcome: all options are distinct, test passed

Test case 3

Description: Check if inserting a single element, it will return a newly ordered Priority Queue with the element included

Input: insert element 22 to Priority Queue [4, 8, 12, 10, 23, 14, 29, 20, 13, 27]

Expected output: [4, 8, 12, 10, 22, 14, 29, 20, 13, 27, 23]

Test outcome: [4, 8, 12, 10, 22, 14, 29, 20, 13, 27, 23], Test passed

Test case 4

Description: Check if deleting an element from a Priority Queue will return a newly ordered Priority Queue with the element deleted.

Input: Delete element 6 from [6, 16, 10, 18, 20, 28, 19, 25].

Expected output: [10, 16, 19, 18, 20, 28, 25] with 6 removed.

Test outcome: [10, 16, 19, 18, 20, 28, 25], test passed

	Test case	**Input**	**Output**	**Method used**
Generate Priority Queue from given List	Check if given a specific list, and compare against a python min heap code which should have the same results	[15, 9, 13, 3, 10, 7, 2, 12]	[2, 9, 3, 12, 10, 13, 7, 15]. Test passed	Manually added [15, 9, 13, 3, 10, 7, 2, 12] To both the Java and Python code and compared the results.
Generate Priority Queue from a non-integer list	Check if the code will still work and output an error message if provided list has an incorrect Data type.	[“a”,”heap”,”2”,”7”,”queue”]	An appropriate an message prompting the user to try again.	Using a list of strings instead of the expected integers.
Insert a specified value	Insert a specified value into the Priority queue to check if it will be placed correctly.	6 is inserted to the Queue [2, 11, 10, 12, 14, 30, 28, 15, 19]	[2, 6, 10, 12, 11, 30, 28, 15, 19, 14]. Test passed	6 is inserted to the list [2, 11, 10, 12, 14, 30, 28, 15, 19] To both the Java and Python code and compared the results.

Delete a value from the Priority Queue	Delete a specified value from the Priority queue to check if it will be removed and the Queue will be reordered accordingly.	Remove 10 from the Queue [3, 8, 10, 21, 29, 24, 23, 28, 25]	[3, 8, 23, 21, 29, 24, 25, 28], Test passed	Remove 10 from the Queue [3, 8, 10, 21, 29, 24, 23, 28, 25] To both the Java and Python code and compared the results.

Test Cases for Binary Search Trees and AVL Trees:

The tests for BST and AVL was done manually to verify whether the output were actually BSTs or AVLs. This was conducted by first taking the answers obtained and questions generated and checking the validity for those outputs manually by using the algorithm. It was found out that those tests passed and this confirming the validity of the BST and AVL trees questions.

References

[1] https://geeksforgeeks.com

[2] Th. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, MIT Press, Cambridge, Massachusetts, 2009.

[3] Mark A. Weiss. 2011. Data Structures and Algorithm Analysis in Java (3rd ed.). Addison Wesley.

[4] https://www.geeksforgeeks.org/binary-search-tree-data-structure/

[5] https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/

[6] https://www.geeksforgeeks.org/binary-search-tree-set-2-delete/

[7] https://www.geeksforgeeks.org/sorted-array-to-balanced-bst/

[8] https://www.geeksforgeeks.org/priority-queue-class-in-java-2/

[9] https://www.geeksforgeeks.org/generating-random-numbers-in-java/

[10] https://www.educative.io/edpresso/how-to-generate-random-numbers-in-java

[12] https://www.journaldev.com/35238/hash-table-in-c-plus-plus