

Faculty of Applied Sciences  
Bachelor of Science in Computing

**COMP490 Final Year Project  
Progress Report**Academic Year 2022/23

|  |  |
| --- | --- |
| Parsons problem generator and solver | |
|  |  |
| Project number: | 19 |
| Student ID: | P-19-0834-5 |
| Student Name: | Jane Liu |
|  |  |
| Supervisor: | Philip Lei |
| Assessor: | Charles Lam |
|  |  |
| Submission Date: | 24/11/2022 |

Declaration of Originality

I, Jane Liu, declare that this report and the work reported herein was composed by and originated entirely from me. This report has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the bibliography.

Text

Description automatically generated with medium confidence

Jane Liu

02/11/2022

Abstract

Parsons problem, a type of programming question reordering mixing up prepared code blocks by dragging and dropping to build predefined answer codes, has been used in introductory programming popularly. However, the existing Parsons problem has never been expanded in higher-level programming courses like Data Structures and Algorithms. Since this course is too complex and abstract for students to write codes directly, it is also vital to use Parsons problem to add preparation steps before they actually begin writing codes by themselves. It cannot just simply change the operating codes with the same structure of Parsons problem due to the difference between these two courses. Therefore, in this project, four concrete types of questions from Data Structures and Algorithms are analysed to show the drawbacks for directly using the traditional one and amelioration aiming at these drawbacks.

Table of Contents

[1 Introduction 7](#_Toc120221924)

[1.1 Objectives 7](#_Toc120221925)

[1.2 Risk Assessment 9](#_Toc120221926)

[1.2.1 Risk 1: The computer used for implementation may break down 9](#_Toc120221927)

[1.2.2 Risk 2: The system does not support different devices 9](#_Toc120221928)

[1.2.3 Risk 3: Users reject this website because of the inferior user interface 9](#_Toc120221929)

[1.3 Summary 10](#_Toc120221930)

[2 Background and Related Work 11](#_Toc120221931)

[2.1 Background 11](#_Toc120221932)

[2.2 Related Work 12](#_Toc120221933)

[3 Completed Work 13](#_Toc120221934)

[3.1 Parsons Problem Analysis 13](#_Toc120221935)

[3.1.1 Object Oriented Programming 13](#_Toc120221936)

[3.1.2 Algorithm Analysis 16](#_Toc120221937)

[3.1.3 Recursion 18](#_Toc120221938)

[3.1.4 Data Structures and Algorithms 19](#_Toc120221939)

[3.2 Balance the degree of difficulty 22](#_Toc120221940)

[3.2.1 Switch Different Difficulty Level 22](#_Toc120221941)

[3.2.2 Pre-scaffold 23](#_Toc120221942)

[3.2.3 Context 23](#_Toc120221943)

[4 On-going and Future Work 24](#_Toc120221944)

[4.1 On-going Work 24](#_Toc120221945)

[4.1.1 Data Modelling 24](#_Toc120221946)

[4.1.2 Implementation of the prototype 24](#_Toc120221947)

[4.2 Gantt Chart 26](#_Toc120221948)

[4.3 Future Work 27](#_Toc120221949)

[5 Conclusion 28](#_Toc120221950)

[References 29](#_Toc120221951)

Table of Figures

[Figure 1: Probability impact matrix – initially (left) and solutions (right) 10](#_Toc120221952)

[Figure 2 Changing method positions does not affect the correctness of codes 14](#_Toc120221953)

[Figure 3: The description of methods in a stack (left) and the corresponding codes (right) 15](#_Toc120221954)

[Figure 4 The traditional Parsons problem (above) and the variation of Parsons problem (below) 16](#_Toc120221955)

[Figure 5: Performance of a heap-based priority queue 18](#_Toc120221956)

[Figure 6 The recursion example code of the Tower of Hanoi 18](#_Toc120221957)

[Figure 7 Steps for solving Fibonacci Sequence 19](#_Toc120221958)

[Figure 8: The array-based stack and the linked-list-based stack 20](#_Toc120221959)

[Figure 9 The ordered array stack code pool (left) and the unordered linked-list stack code pool (right) 21](#_Toc120221960)

[Figure 10 Selection sort (left), insertion sort (middle) and the mixed code pool (right) 22](#_Toc120221961)

[Figure 11 Example for inserting key code fragments 23](#_Toc120221962)

[Figure 12 ER diagram 24](#_Toc120221963)

[Figure 13 Create new question 25](#_Toc120221964)

[Figure 14 Upload predefined solution 25](#_Toc120221965)

[Figure 15 Rearrange the code blocks vertically by dragging and dropping 26](#_Toc120221966)

[Figure 16 Check users' answer and give feedbacks 26](#_Toc120221967)

[Figure 17 Gantt Chart 27](#_Toc120221968)

List of Tables

[Table 1: Table of prioritized risk 9](#_Toc118942147)

# Introduction

In recent years, with the computer industry boom, a growing number of people prefer to major in science in computing at their universities in order to find decent jobs after graduating. However, for some of the learners of this program, the courses are not straightforward, especially some courses having programming practices. To equip students with the programming skills, an advanced learning method, Parsons problem, has been introduced into the introductory programming courses (CS1 [1]). With the drag-and-drop programming style, this method allows the students to reordering the mixing up prepared blocks of codes to build the predefined solutions, which provides students with preparation steps by giving them sterling answers to refer before they need to write codes by themselves. Although Parsons problem gives notable assistance in programming study, it is limited in the programming practices in CS1 and it has never been used in teaching activities in higher-level programming courses such as Data Structures and Algorithms (CS2 [1]). Admittedly, students taking intermediate-level programming courses should be able to write code instead of just rearranging the order of provided answer blocks. However, because of the abstractness and complexity of programming in this course, it is also a challenging task for students to write it by themselves directly. Thus, it is of the essence to introduce Parsons problem to build a “bridge” for students to grow their programming capability. However, since there are some differences between CS1 and CS2, the traditional Parsons problem cannot be applied in CS2 without any adjustment. Therefore, in this project, how to use Parsons problem in new ways to fit the programming exercises in CS2 is discussed.

## Objectives

This project aims to design and build a website for computer usage to let teachers input problems and solutions (in Python), generate Parsons problems according to the inputs, make students solve the generated Parsons problems, and provide feedback for the solutions. The major objectives of this project are as follows:

* Study the existing Parsons problem components (question description, correct code fragments and distractors) and common methods to reduce the difficulty (pre-scaffold and context)
* Design new ideas for using Parsons problem in CS2 according to the concrete questions in exercise of CS2
* Design the system architecture to have a blueprint of the project
* Design and implement a database to store the data about accounts, questions, solution code fragments, distractors, and feedbacks
* Design and implement a web page for teachers to input the questions with corresponding solutions
* Design and implement a function to generate Parsons according to the inputs problem, including cutting the whole python file into code fragments and presenting these code fragments in a random way
* Design and implement a web page showing code pools with code fragments in random orders and structured place with vertical dimension and horizontal dimension
* Design and implement a function for students to drag and drop code fragments from the pools and put them in the structures places in proper order with proper indentation
* Design and implement a function for students to evaluate the code generated by students and give them some feedback

In this project, the Parsons problem will be used to handle the exercises in CS2 to have preparation steps for students to write codes by themselves. The variation of Parsons problem can not only take the advantages of the existing Parsons problem (like improving students’ engagement and reducing students’ cognitive load [2]), but also have some new benefits. To be more specific, the new design can handle the specific concepts only existing in CS2. In addition, it can help students to build excellent habits to solve problems. Last but not least, it can help students to consolidate what they have learned through many different ways.

There are several challenges in this project. Firstly, it needs to study intensively the concrete exercises in CS2 to figure out the possible amelioration of existing Parsons problem. Secondly, it needs to design and develop proper Parsons problem fitting the exercises in CS2. Last challenge is to implement a 2D view for dragging and dropping the code blocks.

## Risk Assessment

This section will describe three risks encountered in the project, illustrate the severity priorities of these three risks, and show the effectiveness of the related solution by comparing the impact and possibility before and after.

### Risk 1: The computer used for implementation may break down

Since the whole project work, including codes and documents, is stored on the computer, it will be a disaster when computer errors occur. All work will disappear in a moment, and the previous effort will go down the drain, which will seriously affect the project's progress.

Solution: Take advantage of online backups such as GitHub or Google Drive to ensure the progress will not be affected when the computer acts up.

### Risk 2: The system does not support different devices

Although this web-based project is designed to be used on computers majorly, some users may try to open it on their tablets. Since the changes in screen sizes between these three kinds of devices, some interfaces of web pages may look too small or too strange, which may bring users an awful experience and reduce their motivation for further use of this website. In that case, it does not seem conducive to the future growth of this project.

Solution: Design a responsive website to fit the size of different devices satisfying users’ needs on various devices.

### Risk 3: Users reject this website because of the inferior user interface

If the user interface is unfriendly, the user may misunderstand some instructions given by this website and repeat some unexpected wrong actions again and again, which wastes a lot of users’ time and may cause users to be impatient and give up easily. And this situation is not beneficial for the long-term development of this project.

Solution: Apply common design principles like Eight Golden Roles to the interfaces. In addition, simulate a user to access some related projects and get some ideas from their design.

To sum up the severe level of these previously mentioned risks, a priority risk table has been shown in Table 1.

Table 1: Table of prioritized risk

|  |  |
| --- | --- |
| Priority | Risk Identifier and Description |
| 1 | Risk 1: The computer used for implementation may break down |
| 2 | Risk 3: Users reject this website because of the inferior user interface |
| 3 | Risk 2: The system does not support different devices |

As shown in Table 2, there are three related risks. Specifically, the first risk has the highest priority since it may redo the whole work, which may significantly delay the development of this project. Subsequently, the third risk has lower priority than the previous risk since the project can still perform when affected by this risk. However, this risk may cause the users to feel confused and fretful, so they may not want to use the website again, which is unsuitable for the popularization of the website. Lastly, the second risk has the lowest priority since this website will be major used on computers, and only a few people using other platforms are affected by this risk.

The probability impact matrixes have been shown in the following two figures to directly show between before and after using solutions.

Table

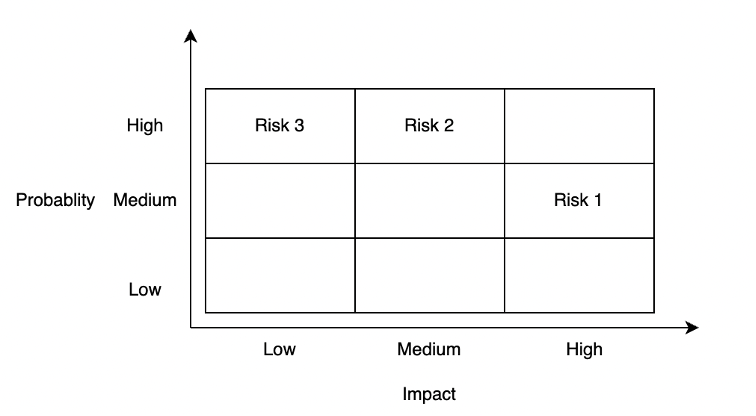
Description automatically generated

Figure 1: Probability impact matrix – initially (left) and solutions (right)

## Summary

This report is organized as follows: Chapter 2 introduces the background and related work of Parsons problem. Chapter 3 presents the problem analysis. Chapter 4 shows the on-going work and future work. Chapter 5 is the conclusion.

# Background and Related Work

This chapter explains the background and related work of Parsons problem. In the background section, the components and structures of Parsons problem are explained. In addition, the brief difference between the existing Parsons problem and the new ones are also explained.

## Background

The Parsons problem is a drag-and-drop-style code-reordering problem. Precisely, in a Parsons problem, a prepared correct, complete solution code of a specific problem is cut into plenty of small code fragments in lines. These code fragments are provided to students in a random order in a code pool with some incorrect or improper ones (distractors). And students should select the correct code fragments from the code pool and drag and drop these code fragments to place them the correct order. During this period, students can get some instruction feedback based on their reordered answers, and they need to improve their answers based on the feedback until they get the same answer as the predefined answer [2].

The Parsons problem contains the following four components:

* **Question descriptions** illustrate the questions in text or activity diagrams. Using activity diagrams in Parsons problem, students can understand the problem’s solution logic before they begin solving it [2].
* **Correct code fragments** are lines in correct solution codes. Since students only need to choose and reorder prepared code fragments instead of writing codes directly, the cognitive load is reduced, so the difficulty of programming questions is discounted [2]. In addition, this design provides magnificent solution examples as reference material for students to analogize some similar questions.
* **Distractors** are some incorrect or improper code fragments mixing with the original correct ones. By comparing with the correct code fragments, students can not only distinguish some common errors but also cultivate good coding habits like using meaningful and conforming naming rules names [2].
* **Feedback** is used to give students improving hints based on students’ solutions. There are two different types of feedback – line-based feedback and execution-based feedback. Line-based feedbacks evaluate students’ solution by comparing them with the prepared solution, while execution-based feedbacks evaluate students’ solution by running codes directly. Both kinds of feedback can provide students with instant feedback to give students some instructions for improving [3].

Besides the contribution of Parsons problems components in reducing difficulties of programming questions, there are two common methods to ease the programming problem further:

* **Pre-scaffold**: it reduces difficulty by providing hints of the organization of solution codes. It can be achieved by indicating some structures like loop structures based on braces or indentations [3]. Moreover, it also can be achieved by giving subgoals [4].
* **Context**: some settled code is provided around the sorting codes [6].

## Related Work

This method has been used in many previous work about programming education. To be more specific, Parsons and Haden developed a web-based tool for generating Parsons problems for introductory programming in BASIC with jumbled distractors, pre-scaffolding, and absolute line-based feedback [2] [3]. Besides, Karavirta et al. developed a Parsons problems project called js-parsons for introductory programming in Python with no distractors, student-scaffolding, and relative line-based feedback [3] [6]. Lastly, Amruth developed Epplets for introductory programming in C+, C#, and Java with distractors and relative line-based feedback [3] [7]. In conclusion, although the previous work provides Parsons problem practice in several programming languages with different structures, these projects have one thing in common – they are only applied for introductory programming, which gives this project room for improvement.

# Completed Work

This chapter explains the completed work, including the new ways of using Parsons problem to fit the exercises in CS2.

## Parsons Problem Analysis

The traditional Parsons problem has shown its advantages in exercises of CS1. However, since there are plenty of differences between the exercises in CS1 and CS2, it is not so proper to apply Parsons problem directly in CS2 without tailored improvement. Thus, some new ideas should be introduced to Parsons problem. The limitation of the traditional Parsons problem and new ideas will be discussed in the following paragraphs by comparing the difference between CS1 and CS2.

Unlike CS1, the codes in CS2 are much more complex and abstract in the following three general aspects. To be more precisely, students are supposed to not only understand the meaning of every line but also be able to organize these single lines into larger blocks (like methods, classes, even algorithms) in order to have a deeper understanding of the features of data structures, the recursive solving method, and the algorithms analysis. In addition, codes are no longer the only points covered in Parsons problem. Some new abstract elements like pseudocode and algorithms analysis should also be involved in exercises of CS2. Finally, high comparing ability is required in CS2 since there are plenty of similar concepts like stacks and queues, different types of trees, and different algorithms to solve the problems.

The difference between the CS1 and CS2 courses mentioned above leads to deficiencies in applying the traditional Parsons problem. To show these limitations in detail and introduce the improvement aiming at these limitations, the exercises in the famous textbook *Data Structures and Algorithms in Python* [1]were analyzed, and four common question types were identified.

### Object Oriented Programming

Object-oriented programming is a programming model based on objects, which includes attributes and methods [2]. In CS1, since all the programming parts are Structured Programming, the object-oriented programming concept has never been involved. However, in CS2, this concept becomes one of the focal points since it is the foundation to build different data structures. But after introducing this concept, Parsons problem for data structure questions cannot have unique answers since the methods in the classes can change their positions without affecting the correctness of the results (for example, Figure 2).

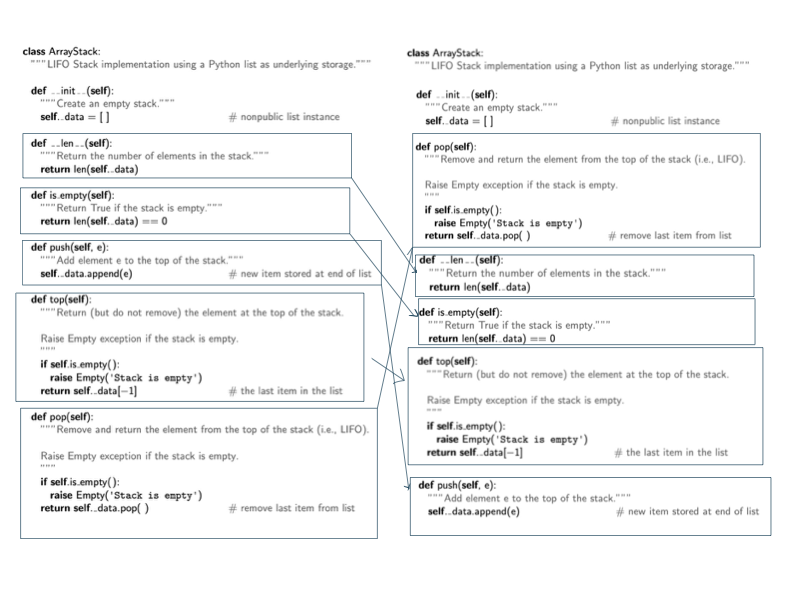


Figure 2 Changing method positions does not affect the correctness of codes

To have a unique answer, some paragraphs describing the functions of the methods should be listed in a specific order in the question description, and students are asked to build methods one by one in a matching order. For example, in Figure 3: The description of methods in a stack (left) and the corresponding codes (right), the left part shows the description of methods of an abstract data type stack in a specific order, and the corresponding methods are implemented in a matching order in the right part. In this way, a unique order of answers is defined by design. Thus, Parsons problem can not only fit the object-oriented programming type of questions but also keep some of its reordering features simultaneously. More importantly, this new design idea enhances students’ understanding of the whole method instead of just every individual line. Further, by this design, students also can have a general idea about what features of a specific abstract data type (ADT) should have [1].

Graphical user interface

Description automatically generated with low confidence

Figure 3: The description of methods in a stack (left) and the corresponding codes (right)

Another way to handle this problem is only to check the availability of methods and the contents of methods but ignore the positions of the methods. This variation of Parsons problem is significantly different from the traditional Parsons problem (Figure 4). Specifically, in the traditional Parsons problem, the interchangeable lines are combined into a larger block to have a unique answer [3]. In this way, it ignores the smaller parts (lines) and focuses on the bigger ones (blocks). By contrast, the new design looks the opposite way compared to the previous design. It ignores the bigger parts (order between different methods) and focuses on the smaller parts (lines inside the methods).

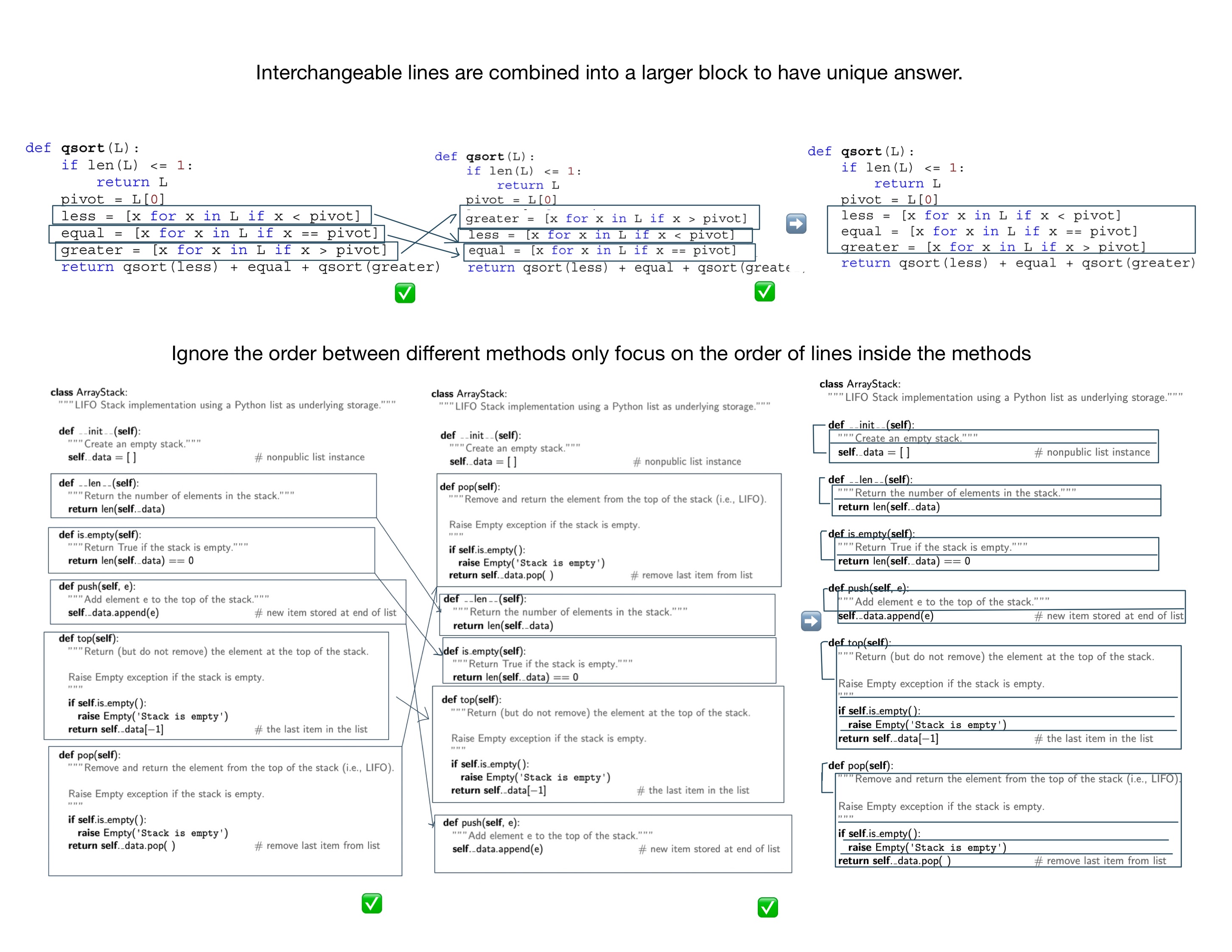


Figure 4 The traditional Parsons problem (above) and the variation of Parsons problem (below)

### Algorithm Analysis

Algorithm analysis, which is a core part of CS2, is to use the running time to evaluate whether a data structure or an algorithm is efficient or not [1]. Since the growth rate of running time as a function of the input size n is vital for algorithm analysis, big O, which shows the significant parts affecting the growth, is introduced in algorithm analysis [1]. Since the only major data structure used in CS1 is array – one of linear data structure, the algorithm analysis in CS1 can be done simply by counting the number of times a statement runs. In other words, the algorithm analysis in CS1 mostly depends on the loop structures in the codes so that there are only four functions (the constant function, the linear function, the quadratic function, and the cubic function and other polynomials) are used in programming. By contrast, the codes in CS2 involves more complicated non-linear data structures such as trees or graphs, which means that students should not only consider the impact of code structures but also non-linear data structures when analyzing the codes. For example, a loop to traverse a balance tree will consume O(log n) steps, not O(n) steps (where n is the number of nodes in the tree). Thus, more big O classes are used commonly in CS2 (the constant function, the logarithm function, the linear function, the N-log-N function, the quadratic function, the cubic function and other polynomials, and the exponential function) [1]. It is strenuous to ascertain which functions are the correct functions to describe the efficiency of the specific data structures and algorithms. Besides the impact of non-linear data structure, the recursion is another reason to make the analyzing experiencing learned from CS1 impractical. It is not straightforward to observe how many times a recursion function done. For these reason, the algorithm analysis parts are always ignored in CS1, so the traditional Parsons problem for CS1 certainty cannot handle this algorithm analysis questions, which is another drawback for using traditional Parsons problem directly in CS2.

To involve the algorithm analysis, only operating the code cannot satisfy the situation anymore. Thus, a new component – comment – is introduced in Parsons problem. To be more specific, the comments are used to provide different big O classes for choosing. Students need to select the correct corresponding big O class comments and insert them into codes to do algorithm analysis of the whole code. Unlike the previous reordering questions, this question is more like a multiple-choice question since students can put the comment wherever they like. Furthermore, this new idea also can be expanded in two ways. One way is to provide not only big O but also the described reasons to get big O. In this way, students can have a deeper understanding of algorithm analysis instead of just guessing big O classes according to their feelings. Another way is to let students match different big O classes with different parts in codes. This way can be used to track the change points of big O classes (like outer for loop and inner for loop), or it can be used to compare different methods in a data structure and let students know the characteristics of a data structure (For example, a heap is quick to get the min value, but it is slow to add nodes, shown in Figure 3).

Table

Description automatically generated

Figure 5: Performance of a heap-based priority queue

### Recursion

Recursion is solving a problem by solving a subproblem with the same structures as the original problem. It is achieved by calling the function itself, passing smaller argument values to build subproblems, finding the answers to subproblems, and using these answers to solve the original problem. Unlike in CS1, the recursion method is called inside instead of outside of the method body, which requires students with deeper understanding in the whole picture of entire methods besides concrete lines in the methods. For this reason, it leads to a dramatic increase in the difficulty of problems, and the traditional Parsons problem cannot significantly reduce complexity. Concretely, the traditional Parsons problem reduces difficulty by providing some code fragments for students to read. However, in the recursion type of question, it is arduous to understand the meaning of every line of code even if the whole code is given in order (like the recursion example code of the Tower of Hanoi in Figure 4 [4]). Thus, the assistance coming from code reading of Parsons problem is notably shrunk, and it is almost like letting students write code directly. That is why the traditional Parsons problem needs to be improved to give students more hints in solving recursion questions.

Text

Description automatically generated

Figure 6 The recursion example code of the Tower of Hanoi

According to the suggestions for designing recursive algorithms provided by the textbook, it is beneficial to find a few specific subproblems which have the same structures as the original problems [1]. To achieve the suggested way, a new pre-scaffold way – organizing in several steps – is introduced into Parsons problem. To be more specific, one recursion question is divided into some steps. The first step is to let students reorder the code of base cases (the end point of every recursion chain [1]). Subsequently, change the values of parameters used in this recursion algorithm to have other subproblems. And in this step, the previous base cases should also be involved. In other words, students need to build these subproblems by calling the method in the base cases. In the following step, students should choose the correct method headers with proper signatures. In the final step, students are required to reorder the actual recursion codes with all the previous codes as references. This new pre-scaffold way can be partially helpful for students to solve the recursion problems. In this new way, students are provided with more hints (like subproblems) than directly solving recursion questions. Thus, they are less like to be all at sea (Figure 7).

Graphical user interface, text, application, email

Description automatically generated

Figure 7 Steps for solving Fibonacci Sequence

### Data Structures and Algorithms

One of the characteristics of code in CS2 is similar to each other. For example, the codes can have the same data structures but are implemented in different ways (like the array-based stack and linked-list-based stack in Figure 5) or different solving algorithms for the same problems (like bubble sort and selection sort for sorting numbers problem). Although it is ok to use the traditional Parsons problem individually for each code, it would be more worthwhile to help students to compare these similar codes and consolidate the difference and similarities between these codes. Therefore, the students can distinguish them and have a better understanding.

Table

Description automatically generated

Figure 8: The array-based stack and the linked-list-based stack

The transfer learning method is used to compare the same data structures implemented in different ways. In other words, two different codes with some common parts are provided, and students need to choose the common parts from one of the codes using these common parts to build another one. In this way, students can understand new knowledge by transforming what they have learned before. Taking the two implementation ways of a stack as an example , students can drag and drop the method headers in an array-based stack as some components to build a linked-list-based stack. In such a manner, students can have a general idea about the common parts (the methods supported in a stack abstract data type) and what differences are (implementation details caused by array structure or linked list structure). Besides, the completed ordered array-based stack codes also can be hints to support students in building the linked-list-based stack Figure 9.

Graphical user interface, text, application, email

Description automatically generated

Figure 9 The ordered array stack code pool (left) and the unordered linked-list stack code pool (right)

To compare different algorithms of the same question, two algorithms are mixed by using distractors, and students are supposed to distinguish and order each of them from the mixing code pool. More specifically, unlike traditional distractors, which are all used to show some incorrect or improper code [5], a group of correct, complete codes of an algorithm is introduced as distractors to mix with another algorithm. In other words, the jumbled code fragment pool includes two different correct algorithms, and students should pick up, rearrange, and submit the code fragments in these two algorithms separately. This setting method is significantly applicable in CS2 since many questions have this characteristic – one question with several solving algorithms, for example, bubble sort, selection sort, insertion sort for sorting numbers, and breadth-first search, depth-first search for searching. In this way, it can help students to distinguish similar algorithms in the same categories mentioned before. It is worthwhile when students have learned more than one algorithm and begin to use them mostly because of blurry memory Figure 10.

Text

Description automatically generated

Figure 10 Selection sort (left), insertion sort (middle) and the mixed code pool (right)

## Balance the degree of difficulty

The critical function of Parsons problem is to add a preparation stage before students begin to write codes. This preparation stage is achieved mainly by reducing difficulty caused by directly writing codes. In other words, one of Parsons's key issues is to find a balance point in the degree of difficulty, which is more accessible than writing codes from scratch but still valuable to practice. To fulfill this demand, three different methods (selecting difficulty level, pre-scaffold, and content) are introduced in the following paragraph.

### Switch Different Difficulty Level

In the traditional Parsons problem, whether pre-scaffold or not, the problem is always predefined with a fixed difficulty level, which means that students cannot ask for further help with additional hints if they meet problems in solving this Parsons problem. Since the codes in CS1 are pretty simple, this problem may not happen frequently and may not affect students solving problems. However, due to the increase in the complexity of the codes in CS2, this problem begins to become a huge issue. Multiple difficulty levels are set to handle this issue, and students are allowed to switch to a more accessible version of this question halfway. For example, three different difficulty levels are set. Specifically, Level 0 is to just cut codes in lines and allow students to reorder them. Subsequently, in Level 1, the headers of methods and predefined description comments are placed in the correct positions for hints. In Level 2, the codes are grouped by methods so that students can only reorder smaller parts of codes for every method instead of selecting from the whole mixing codes pool. In Level 3, some codes in methods are fixed as context and only some core parts need to be reordered. If students find the present version of Parsons problem too difficult, they can change to a more accessible version.

### Pre-scaffold

Pre-scaffolding is to give students some structures for hints. This kind of method to handle the difficulty has already been used in the existing Parsons problem, like providing loop structures. This method is also used in CS2 courses but with some extension. To be more specific, pre-scaffolding can be achieved by separating codes by different methods like the example shown before. In addition, pre-scaffolding also can be achieved by separating codes into different steps, like the example shown in recursion questions. Finally, this method also can be achieved by separating codes according to different subgoals helping students to divide a vast, complex problem into more minor simple questions [5]. A hint for each subgoal is provided as comment.

### Context

Context is used to provide some ordered codes as information and let students only reorder some smaller parts in the middle of these ordered codes. Context can be done a step further, which means that students only need to insert some key code fragments into the correct location of the whole code file. And to keep the question valuable, some wrong codes are also provided to confuse students. This kind of question is extra valuable for iterator and recursion questions (Figure 11).

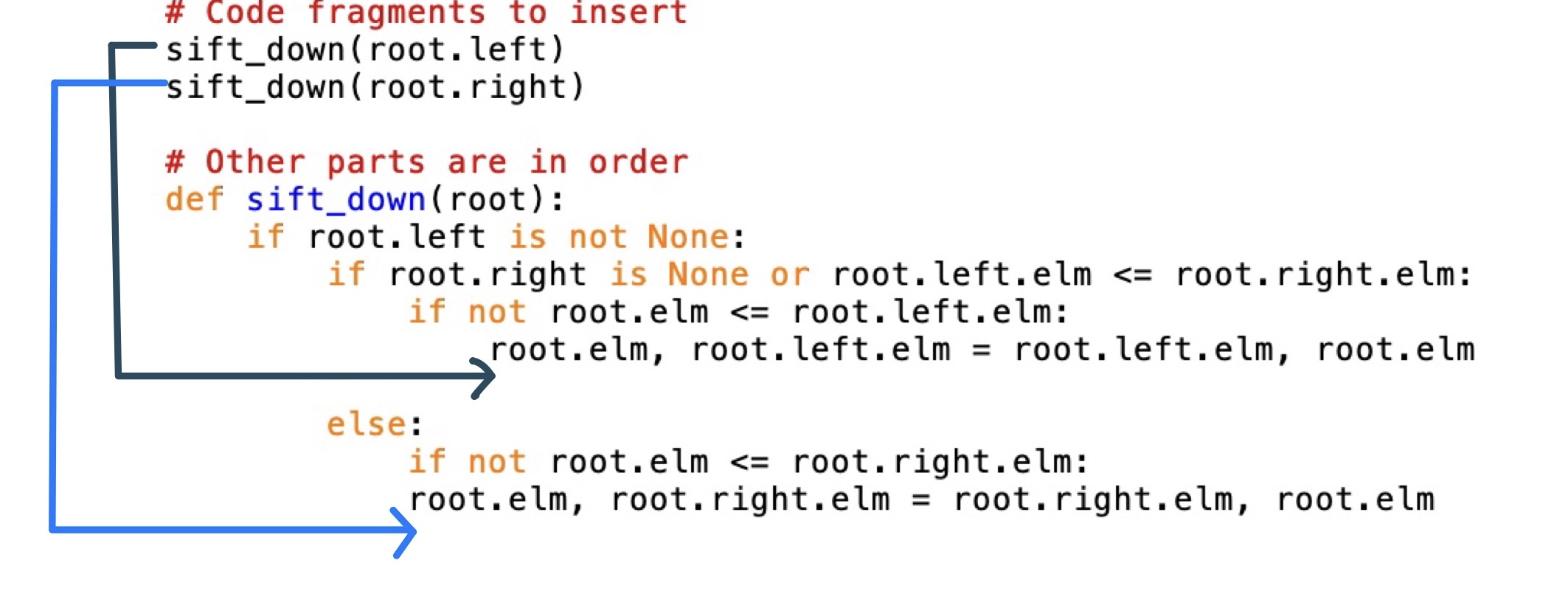


Figure 11 Example for inserting key code fragments

# On-going and Future Work

In this chapter, the on-going and future work are explained. In the on-going work, data modelling and the implementation of the prototype are shown. Subsequently, the Gantt chart is shown in the second part. Finally, a draft idea about what to do to complete the project is explained.

## On-going Work

In this part, the two kinds of on-going work are shown below.

### Data Modelling

To achieve the new design mentioned in Chapter 3, the database is designed as the ER diagram shown below (Figure 6). Since the present database design may be updated in the future, its detailed explanation will be omitted here and shown in the final report.

Diagram

Description automatically generated

Figure 12 ER diagram

### Implementation of the prototype

Because of the time limitation, only a prototype to achieve the basic functions of Parsons problem is implemented. To be more specifically, the prototype can allow users to create a new question with some description (Figure 13). Subsequently, users can upload the predefined answer (Figure 14). In the next step, users can rearrange the code blocks vertically by dragging and dropping (Figure 15). In the last step, the project can check the users’ answer and give some feedback (Figure 16). The 2D structure for putting code blocks and the new functions for exercise in CS2 mentioned in Chapter 3 will be implemented in the future.

Graphical user interface, application

Description automatically generated

Figure 13 Create new question

Graphical user interface, application

Description automatically generated

Figure 14 Upload predefined solution

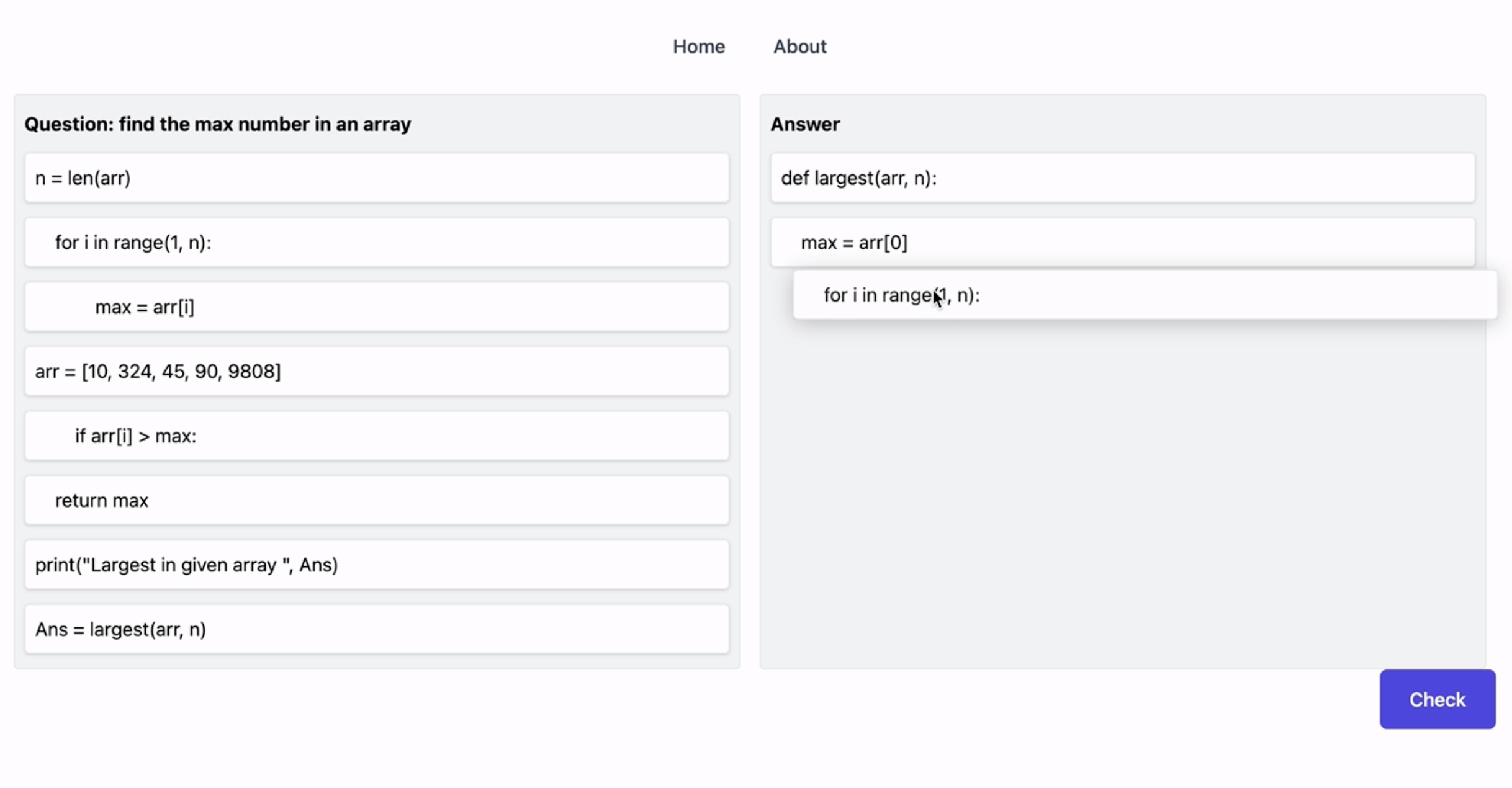


Figure 15 Rearrange the code blocks vertically by dragging and dropping

Graphical user interface, application

Description automatically generated

Figure 16 Check users' answer and give feedbacks

## Gantt Chart

The Gantt Chart is shown below (Figure 17).



Figure 17 Gantt Chart

## Future Work

In the future, the following work needs to be done to finish this project:

* The database needs to be checked and be updated
* Implement the function of inputting and showing multiple solutions
* Implement the function of altering the difficulty level
* Implement the function of handling comments, pseudocodes
* Implement more beautiful user interfaces

# Conclusion

In conclusion, this semester, I analysed Parsons problem in detail and found new ways to use Parsons problem to fit the exercise in CS2. In the future, I will focus on implementation to achieve these new functions.

References

[1] M. Hertz, “What Do "CS1" and "CS2" Mean? Investigating Differences in the Early Courses,” in Proceedings of the 41st ACM Technical Symposium on Computer Science Education, Milwaukee, 2010.

[2] D. Parsons and P. Haden, "Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses," in Australian Computer Society, Inc., Hobart, 2006.

[3] Y. Du, A. Luxton-Reilly and P. Denny, “A Review of Research on Parsons Problems,” in Proceedings of the Twenty-Second Australasian Computing Education Conference, 2020.

[4] P. Denny, A. Luxton-Reilly and B. Simon, “Evaluating a new exam question: Parsons problems,” in Proceedings of the ACM Workshop on International Computing Education Research, 2008.

[5] B. B. Morrison, L. E. Margulieux, B. Ericson and M. Guzdial, “Subgoals Help Students Solve Parsons Problems,” in Proceedings of the 47th ACM Technical Symposium on Computing Science Education, Memphis, 2016.

[6] P. Ihantola and V. Karavirta, "Two-Dimensional Parson's Puzzles: The Concept, Tools, and First Observations," Journal of Information Technology Education: Innovations in Practice, vol. 10, pp. 119-132, 2011.

[7] A. Kumar, “Epplets: A Tool for Solving Parsons Puzzles,” in Proceedings of the 49th ACM Technical Symposium on Computer Science Education, New York, 2018.

[8] M. T. Goodrich, R. Tamassia and M. H. Goldwasser, Data Structures and Algorithms in Python, Hoboken: Wiley, 2018.

[9] “Wikipedia of Object-oriented programming,” [Online]. Available: https://en.wikipedia.org/wiki/Object-oriented\_programming.

[10] “Python Program for Tower of Hanoi,” GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/python-program-for-tower-of-hanoi/.