澳門理工大學
Universidade Politécnica de Macau
Macao Polytechnic University

# Parsons Problem Generator and Solver

*Student: Jane Liu*
*Supervisor: Philip Lei*
*Assessor: Charles Lam*
*Bachelor of Science in Computing (2022-2023)*
*Macao Polytechnic University*
*Email: p1908345@mpu.edu.mo*

## ABSTRACT

Parsons problem has been used as a convenient tool in introductory programming courses to equip students for writing codes. However, intermediate students may also have problems in directly writing codes when taking some higher-level courses, therefore, Parsons problem is also needed in these teaching activities.

This project attempts to build an application of Parsons problem in website form for assisting students in learning one of higher-level programming courses – Data Structures and Algorithms. Based on the analysis of concrete problems in this course, more flexible question types are expanded in this project, including splitting into multiple steps, algorithm comparison and algorithm analysis. In addition, more ways to handle difficulty level is applied, and switching difficulty level is allowed.

The results of this project show that it can provide students with help in familiarizing themselves with codes involved in this course. And experiment will be done in the future to verify the effectiveness.

## INTRODUCTION

Parsons problem is a type of programming question to let students drag and drop to reorder the mixing up prepared blocks of codes to build the unique predefined solutions. This type of question has been applied in introductory programming courses (CS1) popularly. However, the application scope of Parsons problem has never been expanded to higher-level course such as Data Structures and Algorithms (CS2). Admittedly, students taking this course should be able to write some codes instead of just relying on building with predefined code blocks. But because of the abstractness and complexity of this course, it is also challenging for students to write codes by themselves directly, and it is of the essence to introduce Parsons problem as a preparation stage for students to equip themselves for writing codes in the future.

This project aims to build a website to apply Parsons problem in CS2. In this project, not only the common functions of traditional Parsons problem should be implemented (including inputting problems and solutions in Python, generating Parsons problems, solving Parsons problems and giving feedbacks), but also some changes should be made to adapt the specific exercise in CS2 (more different types of questions and more ways to handle difficulty levels).

## CONCLUSIONS

In conclusion, this project explored the possible application of Parsons problem in Data Structures and Algorithms. The new forms of Parsons problem that were developed in this project can support some situations not involved in introductory programming, including different property codes, the coexistence of lines and blocks, and the existence of multiple solutions. These new forms can also handle the difficulty level in different ways.

The project's main contribution is to provide a new technical tool that can help students learn data structures and algorithms more effectively. It can not only improve students' engagement and reducing students' cognitive load as the traditional Parsons problem, but also can help students to consolidate knowledge though comparing similar concepts.

In the future, experiments will be conducted to compare the effectiveness of this tool with traditional learning methods. In addition, questionnaires will be sent to students and lecturers to collect their opinions on improving system.

## PROBLEM ANALYSIS

There are plenty of differences between the exercise in CS1 and CS2, it is not so proper to apply Parsons problem directly in CS2 without tailored improvement. To illustrate the difference in detail, four kinds of concrete questions from exercises in CS2 are analyzed.

**Object-oriented programming:** this model is not involved in CS1, but it is the foundation to implement different data structures. Because of involving this mode, the uniqueness of correct answer is broken since the order of methods in a class can be changed without affecting correctness. To handle this problem, the project should only check the availability and the contents of methods but ignore the positions of the methods.

**Algorithm analysis:** this concept is first introduced in CS2 not mentioned in CS1. To help students to learn algorithm analysis, comments is introduced as options for choosing proper big O classes to describe codes.

**Recursion:** the recursion method requires students with deeper understanding in the whole picture of entire methods besides concrete lines like CS1. For this reason, the traditional Parsons problem cannot reduce the difficulty as usual through providing code reading. To give students some ideas, a recursion question can be divided into several steps to let students build from subproblem to the original recursion codes.

**Comparison:** there are some similar codes in CS2, for example the same data structure with different implementation or different algorithm for solving the same problems. Although it is ok to use the traditional Parsons problem individually for each one, it would be more worthwhile to have ways to help students compare similar codes and consolidate the difference and similarities between these codes. There are two types of questions to have comparison – combine codes from two pools into one pool and split two solutions from one code pool.

## HOLISTIC DESIGN

Questions can be summarized as five different types (Same question can have multiple versions with different question types. Students can switch to easier versions.):

- **Traditional:** the most basic Parsons problem – only support rearranging jumbled predefined codes and getting feedback about the reordered answers. This type of Parsons problem is used in CS1.
- **Multiple Step:** this type of Parsons problem divides the original codes into several steps to let students solve complex problems by solving each small part. It can be used in recursion problem or be used as pre-scaffold
- **Context:** some ordered codes are provided as hints. This type of question can be used to handle the difficulty.
- **Algorithm Comparison:** this type of question is used to compare similar algorithms. In this type of question, students split two solutions from one group of jumbled codes.
- **Algorithm Analysis:** this type of question can introduce the time complexity of algorithm analysis by using big O class comments.

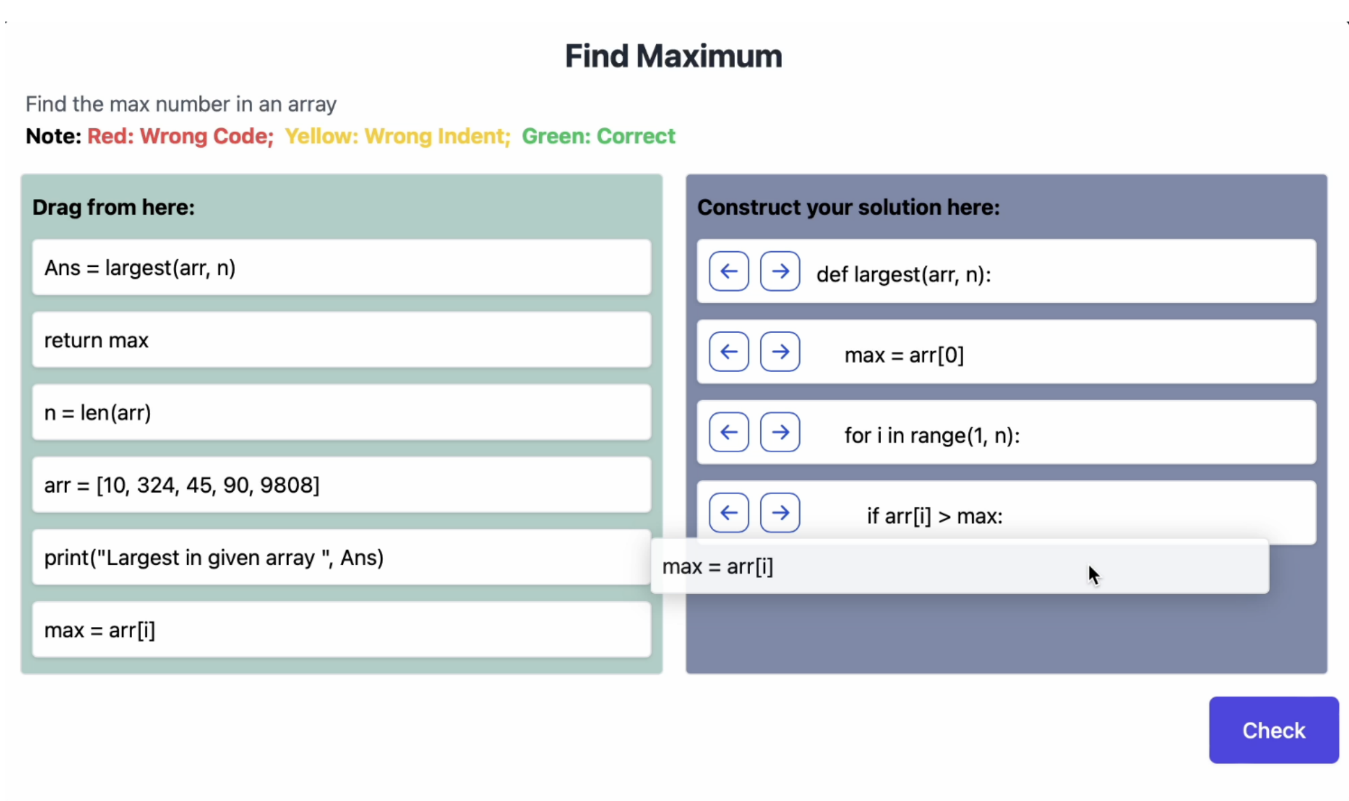
## RESULTS

Figure 1 shows the question page of traditional Parsons problem. Students need to drag the jumbled codes from left code pool, drop the codes in the right code pool in order, and push arrow buttons to set proper indent of codes. After finishing, the codes are marked as different background colors to give students further instruction.

Figure 3 shows the question page of Multiple Step Parsons problem. Compared to the traditional Parsons problem, this form supports breaking a complex problem into many smaller problems. It gives students hints by labeling the number of steps of different codes.

Figure 2 shows the question page of Parsons problem with context. Compared to the traditional Parsons problem, this form provides partially ordered code and placeholders in the initialization phase. This format allows students to get more hints by reading the code in context, thus reducing the difficulty of the problem.

Figure 4 shows the question page of Algorithm Comparison. Compared to the traditional Parsons problem, students need to split two algorithms mixed in left code pool to middle and right code pool separately. This form is worthwhile for students to distinguish similar algorithms and prevent students from using them mostly because of blurry memory.


Find Maximum

Find the max number in an array
**Note:** Red: Wrong Code; Yellow: Wrong Indent; Green: Correct

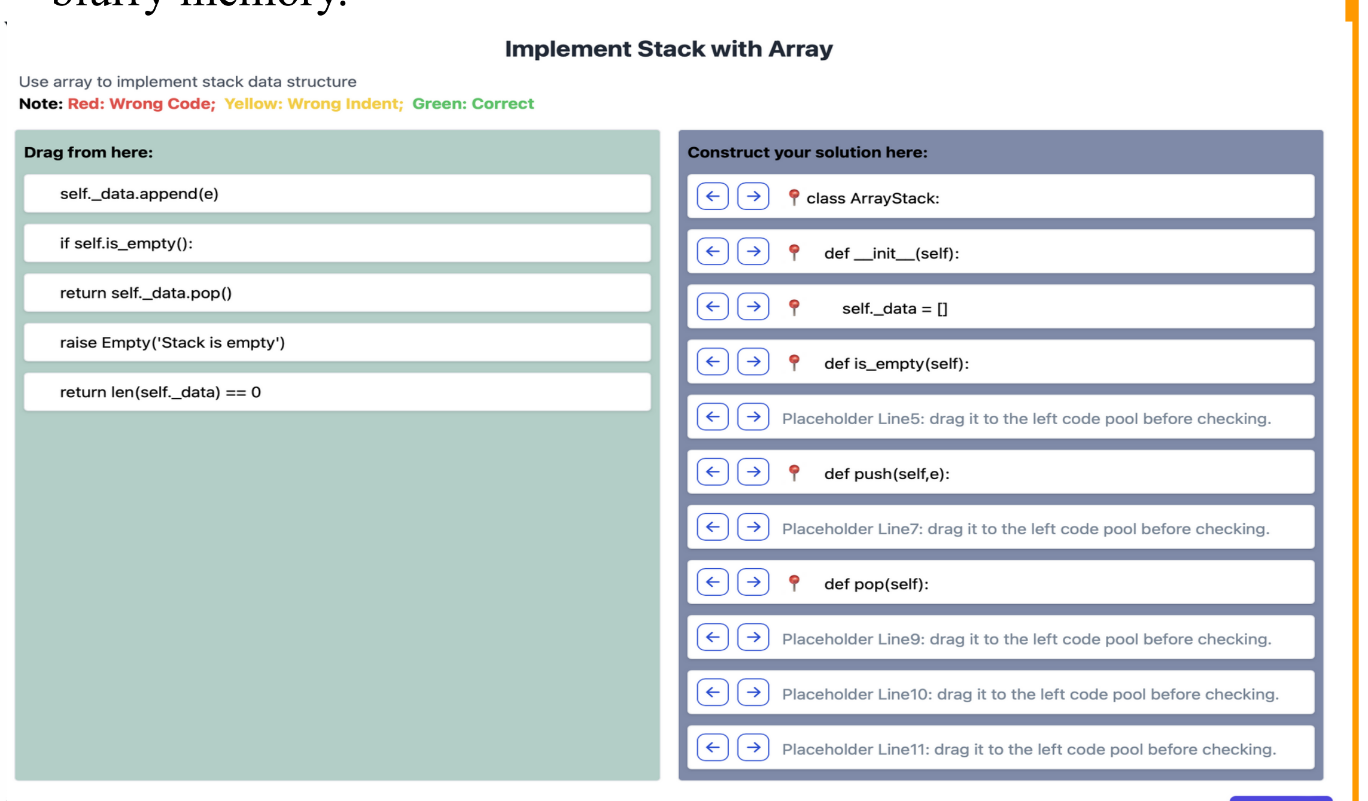
**Drag from here:**

```
Ans = largest(arr, n)
return max
n = len(arr)
arr = [10, 324, 45, 90, 9808]
print("Largest in given array ", Ans)
max = arr[i]
```

**Construct your solution here:**

```
def largest(arr, n):
    max = arr[0]
    for i in range(1, n):
        if arr[i] > max:
max = arr[i]
```

**Figure 1** traditional example: find maximum (ongoing)


Implement Stack with Array

Use array to implement stack data structure
**Note:** Red: Wrong Code; Yellow: Wrong Indent; Green: Correct

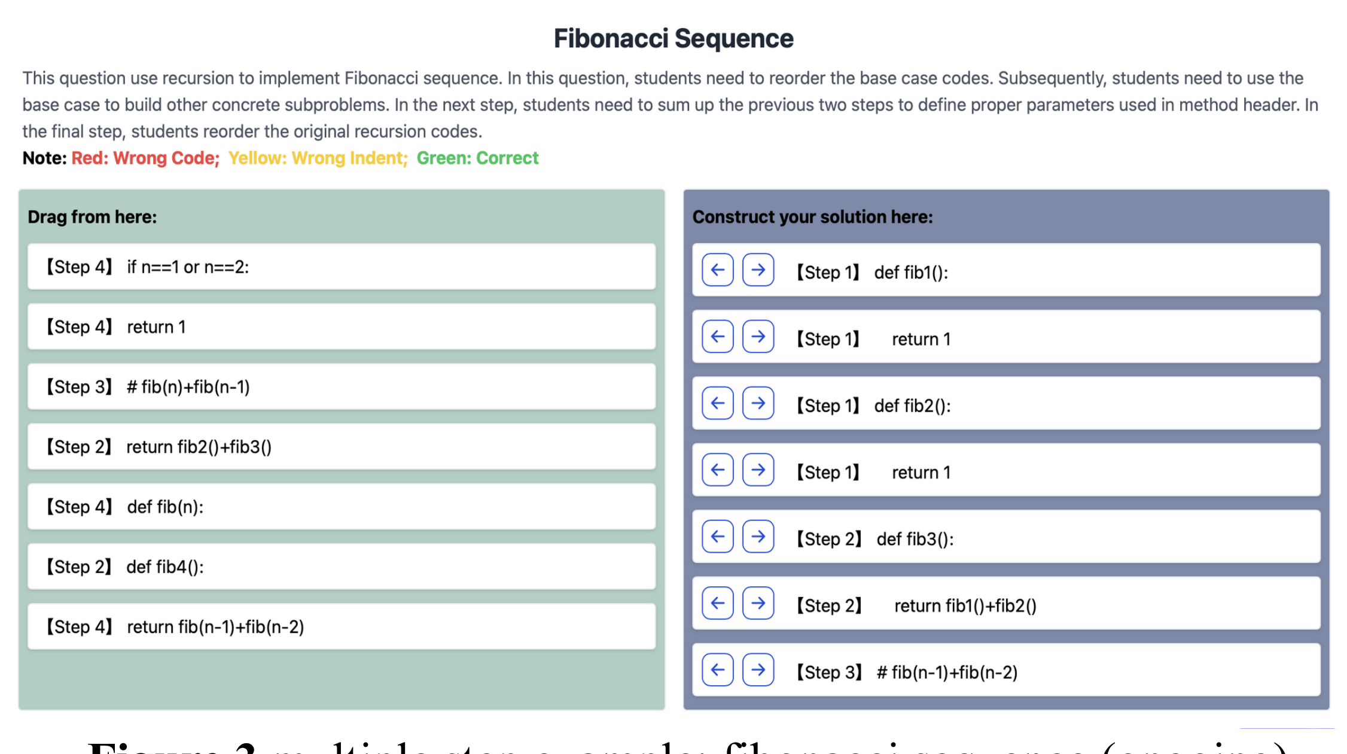
**Drag from here:**

```
self._data.append(e)
if self.is_empty():
return self._data.pop()
raise Empty('Stack is empty')
return len(self._data) == 0
```

**Construct your solution here:**

```
class ArrayStack:
    def __init__(self):
        self._data = []
    def is_empty(self):
Placeholder Line5: drag it to the left code pool before checking.
    def push(self,e):
Placeholder Line7: drag it to the left code pool before checking.
    def pop(self):
Placeholder Line9: drag it to the left code pool before checking.
Placeholder Line10: drag it to the left code pool before checking.
Placeholder Line11: drag it to the left code pool before checking.
```

**Figure 2** context example: implement stack with array (init)


Fibonacci Sequence

This question use recursion to implement Fibonacci sequence. In this question, students need to reorder the base case codes. Subsequently, students need to use the base case to build other concrete subproblems. In the next step, students need to sum up the previous two steps to define proper parameters used in method header. In the final step, students reorder the original recursion codes.
**Note:** Red: Wrong Code; Yellow: Wrong Indent; Green: Correct

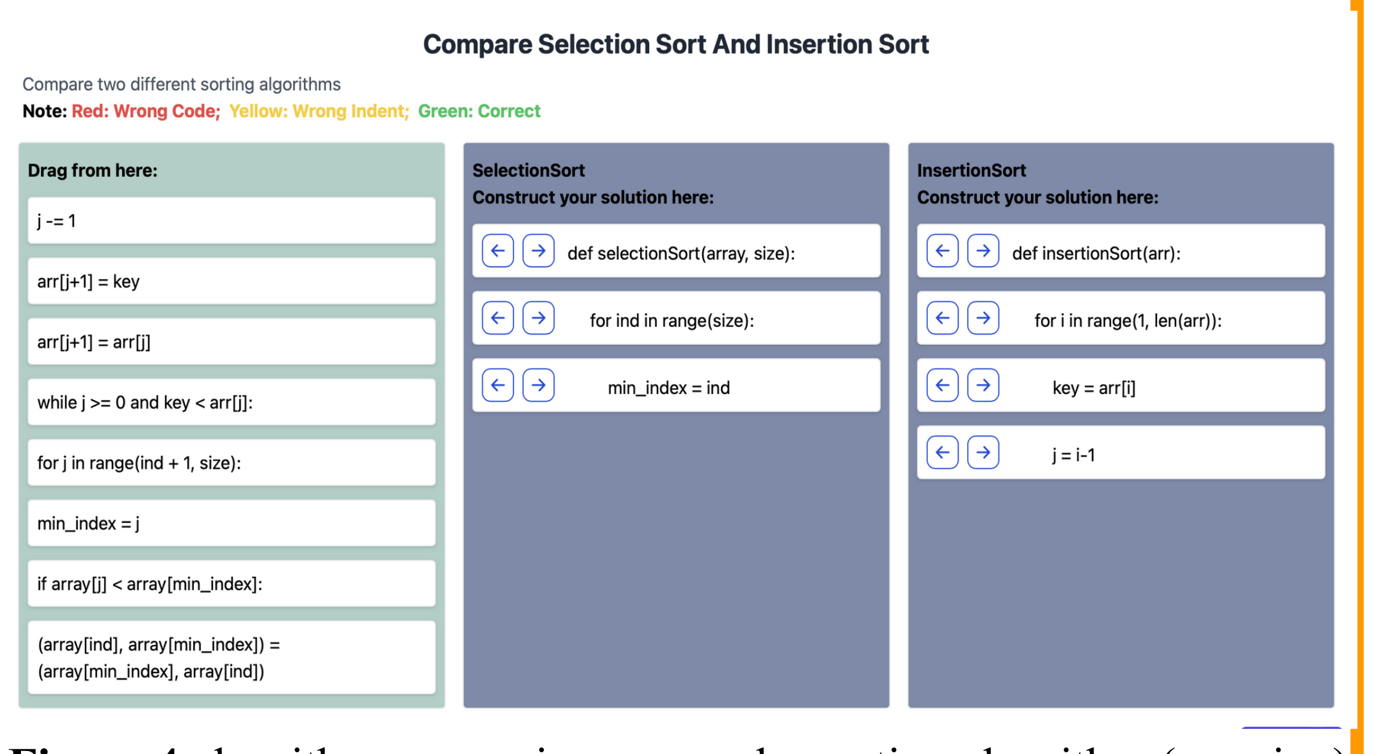
**Drag from here:**

```
【Step 4】 if n==1 or n==2:
【Step 4】 return 1
【Step 3】 # fib(n)+fib(n-1)
【Step 2】 return fib2()+fib3()
【Step 4】 def fib(n):
【Step 2】 def fib4():
【Step 4】 return fib(n-1)+fib(n-2)
```

**Construct your solution here:**

```
【Step 1】 def fib1():
【Step 1】     return 1
【Step 1】 def fib2():
【Step 1】     return 1
【Step 2】 def fib3():
【Step 2】     return fib1()+fib2()
【Step 3】 # fib(n-1)+fib(n-2)
```

**Figure 3** multiple step example: fibonacci sequence (ongoing)


Compare Selection Sort And Insertion Sort

Compare two different sorting algorithms
**Note:** Red: Wrong Code; Yellow: Wrong Indent; Green: Correct

**Drag from here:**

```
j -= 1
arr[j+1] = key
arr[j+1] = arr[j]
while j >= 0 and key < arr[j]:
for j in range(ind + 1, size):
min_index = j
if array[j] < array[min_index]:
(array[ind], array[min_index]) =
(array[min_index], array[ind])
```

**SelectionSort**
**Construct your solution here:**

```
def selectionSort(array, size):
    for ind in range(size):
        min_index = ind
```

**InsertionSort**
**Construct your solution here:**

```
def insertionSort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
```

**Figure 4** algorithm comparison example: sorting algorithm (ongoing)