UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 2

Algorithm Analysis and Flowchart

Submitted by:
Balaoro, Judge Wayne B.

Instructor:
Engr. Maria Rizette H. Sayo

July 26, 2025

I. Objectives

Introduction

Data structure is a systematic way of organizing and accessing data, and an algorithm is a step-by-step procedure for performing some task in a finite amount of time. These concepts are central to computing, but to be able to classify some data structures and algorithms as “good,” we must have precise ways of analyzing them.

This laboratory activity aims to implement the principles and techniques in:

- Writing a well-structured procedure in programming
- Writing algorithm that best suits to solve computing problems to improve the efficiency of computers
- Convert algorithms into flowcharting symbols

II. Methods

A. Explain algorithm and flowchart

B. Write algorithm to find the result of equation: $f(x) = \begin{cases} -x, & x<0 \\ x, & x \geq 0 \end{cases}$ and draw its flowchart

C. Write a short recursive Python function that finds the minimum and maximum values in a sequence without using any loops

III. Results

A.

An algorithm is a step-by-step procedure or set of rules to solve a problem or perform a computation. It's a well-defined sequence of instructions that takes some input and produces an output. Algorithms can be expressed in natural language, pseudocode, or programming languages. Meanwhile, a flowchart is a graphical representation of an algorithm. It uses different shapes (like rectangles, diamonds, and arrows) to represent different types of operations (processes, decisions, inputs/outputs) and the flow between them. Flowcharts help visualize the logic and flow of an algorithm.

B.

Algorithm:

1. Start
2. Input a value for x
3. If x < 0, then set f = -x
4. Else, set f = x
5. Output the value of f
6. Stop

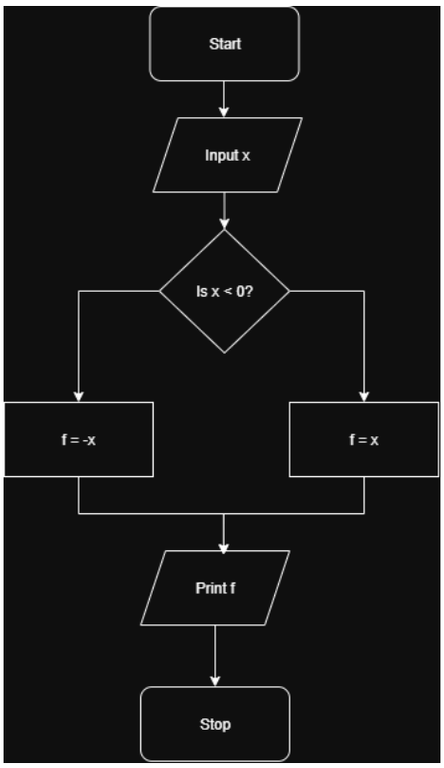

Figure 1. Flowchart

The flowchart shows the steps to find the absolute value of a number. First, it asks you to enter a number. Then, it checks if the number is less than zero. If it is, the flowchart changes the number to its positive form by multiplying by -1. If it is not less than zero, it keeps the number the same.

Finally, it shows the result and ends. This helps to always get a positive number, no matter if the input was negative or positive.

C.

```python
def find_min_max(sequence, index=0, current_min=None, current_max=None):
    if index == len(sequence):
        return (current_min, current_max)

    if current_min is None and current_max is None:
        current_min = current_max = sequence[index]

    current_value = sequence[index]
    new_min = current_value if current_value < current_min else current_min
    new_max = current_value if current_value > current_max else current_max

    return find_min_max(sequence, index + 1, new_min, new_max)

sequence = [5, 2, 9, 1, 7]
min_val, max_val = find_min_max(sequence)
print(f"Minimum: {min_val}, Maximum: {max_val}")
```

```
Minimum: 1, Maximum: 9
```

Figure 2. Source Code

This python source code is a function that looks through a list of numbers to find the smallest and largest values. It checks each number one by one, compares it with the smallest and largest found so far, and updates these values if needed. When it reaches the end of the list, it returns the smallest and largest numbers it found. This way, the code helps find the minimum and maximum numbers in the list.

IV. Conclusion

This laboratory activity successfully demonstrated the fundamental concepts of algorithm design and flowchart representation. By analyzing the given piecewise function and implementing a recursive solution to find minimum and maximum values, the exercise reinforced the importance of structured problem-solving in computer science. The algorithm for f(x) effectively handled conditional logic, while the recursive Python function showcased how iteration can be replaced with recursion for sequence analysis. Flowcharts provided a clear visual representation of control flow, enhancing comprehension of algorithmic logic. These skills are essential for optimizing computational efficiency and developing well-organized code. Moving forward, mastering these techniques will enable more complex problem-solving and efficient program design in future programming tasks.

References

[1] Co Arthur O.. “University of Caloocan City Computer Engineering Department Honor Code,” UCC-CpE Departmental Policies, 2020.