

**Complex Computing Problem (CCP) - Recurrence Relation Solver**

Design and Analysis of Algorithms



April 20, 2025

Department of computer science

University of Engineering & Technology, Lahore (New Campus)

A logo of a university of engineering and technology

AI-generated content may be incorrect.

**Submitted by:**

Muhammad Sami 2023-cs-656

Hassan Farooq 2023-cs-666

Hamza Mehmood 2023-cs-678  
**Submitted to:**  
 Maam Drakhshan Bokhat

## 1. Introduction

The Complex Computing Problem (CCP) solver is a Python application designed to solve recurrence relations using various mathematical methods. This document outlines the flow and structure of the program, explaining how the different components interact.

## 2. Program Structure

The main component of the CCP solver is the RecurrenceRelationSolver class, which encapsulates all the logic needed to parse, analyze, and solve recurrence relations. The program follows a logical flow:

1. User Input - Collect information about the recurrence relation
2. Parsing - Convert the string representation into a structured format
3. Analysis - Determine the appropriate solution method
4. Solution - Apply mathematical techniques to solve the relation
5. Output - Present the solution with optional visualization

## 3. Recurrence Relation Types

The program handles three main types of recurrence relations:

### 3.1 Divide and Conquer

Format: T(n) = aT(n/b) + f(n)

This type appears in algorithms that divide a problem into a subproblems of size n/b each.  
Examples: Binary Search, Merge Sort, QuickSort

### 3.2 Decrease and Conquer

Format: T(n) = T(n-d) + f(n)

This type appears in algorithms that reduce the problem size by a constant amount.  
Examples: Linear Search, Selection Sort

### 3.3 Complex Divide and Conquer

Format: T(n) = T(n/b₁) + T(n/b₂) + ... + f(n)

This type appears in algorithms with multiple different-sized subproblems.  
Examples: Some specialized divide-and-conquer algorithms

## 4. Solution Methods

### 4.1 Master Theorem

The Master Theorem provides a quick way to solve divide-and-conquer recurrence relations of the form T(n) = aT(n/b) + f(n).

It has three cases:  
- **Case 1**: If f(n) = O(n^c) where c < log\_b(a), then T(n) = Θ(n^log\_b(a))  
- **Case 2**: If f(n) = Θ(n^c) where c = log\_b(a), then T(n) = Θ(n^c log n)  
- **Case 3**: If f(n) = Ω(n^c) where c > log\_b(a), then T(n) = Θ(f(n))

### 4.2 Substitution Method

This method involves guessing a solution and using induction to verify it's correct.

Steps:  
1. Guess the form of the solution  
2. Use mathematical induction to verify the guess  
3. Solve for any constants in the solution

### 4.3 Iteration Method

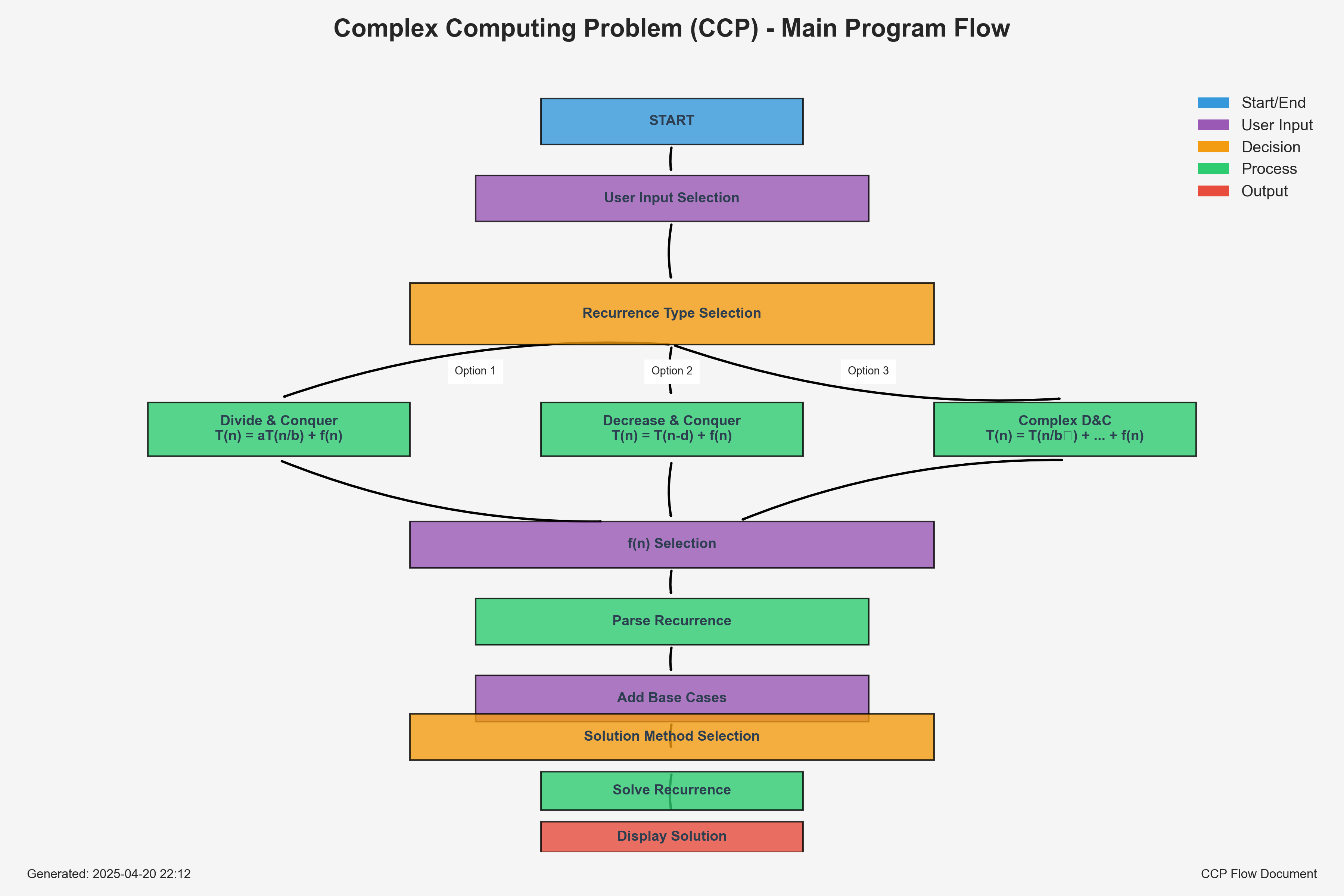
This method involves expanding the recurrence relation and identifying a pattern.

Steps:  
1. Repeatedly substitute the recurrence into itself  
2. Express as a sum of terms  
3. Simplify the sum to obtain a closed form

## 5. Program Flow Diagrams

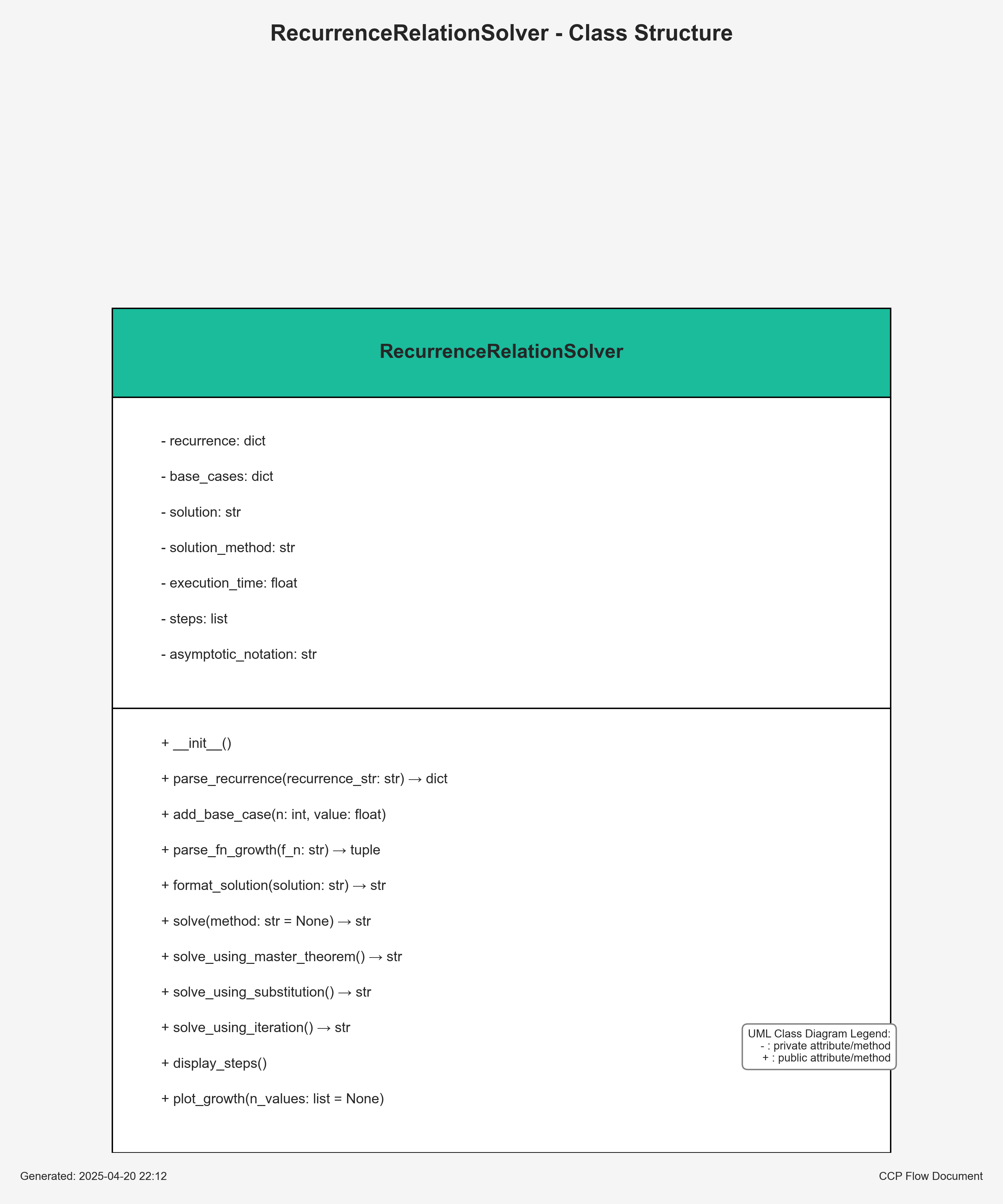
The program flow is illustrated in the following diagrams:

### 5.1 Main Program Flow



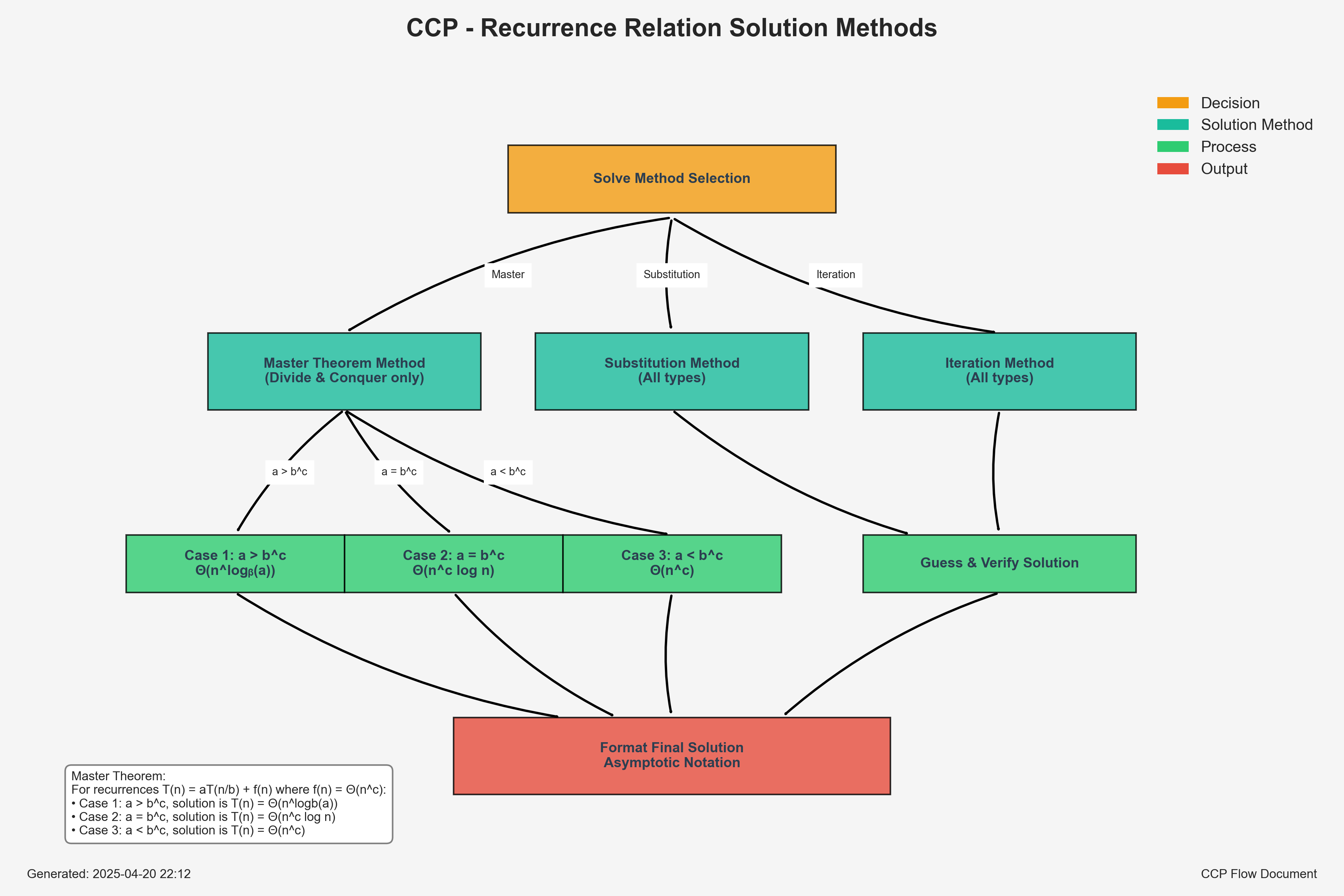
*Figure: Main Program Flow*

### 5.2 Class Structure



*Figure: Class Diagram*

### 5.3 Solution Methods



*Figure: Solution Methods*

## 6. Examples

### Example 1: Binary Search - T(n) = T(n/2) + 1

* Type: Divide and Conquer
* Method: Master Theorem (Case 2)
* Solution: T(n) = Θ(log n)

### Example 2: Merge Sort - T(n) = 2T(n/2) + n

* Type: Divide and Conquer
* Method: Master Theorem (Case 2)
* Solution: T(n) = Θ(n log n)

### Example 3: Linear Search - T(n) = T(n-1) + 1

* Type: Decrease and Conquer
* Method: Iteration
* Solution: T(n) = Θ(n)

## 7. Conclusion

The CCP solver provides a comprehensive tool for analyzing and solving recurrence relations, which are fundamental in algorithm analysis. By understanding the program flow documented here, users can effectively utilize the solver for a wide range of algorithmic complexity analyses.