Algorithm Visualizer Using C++ and Python

A project to visualize recursion and backtracking using a C++ execution engine and a Python-based visualization layer.

1. Introduction

Recursion and backtracking are difficult to understand because their execution flow is hidden inside the call stack. This project visualizes that hidden flow step by step.

2. Objectives

- Visualize recursion and backtracking
- Show stack push and pop operations
- Separate algorithm logic from visualization

3. Algorithms Implemented

- Binary String Generation
- Subset Generation
- Combination Generation
- Permutation Generation

4. Architecture

C++ Layer: Implements algorithms and prints structured execution steps.
Python Layer: Reads output and visualizes execution using indentation and animation.

5. Sample Input

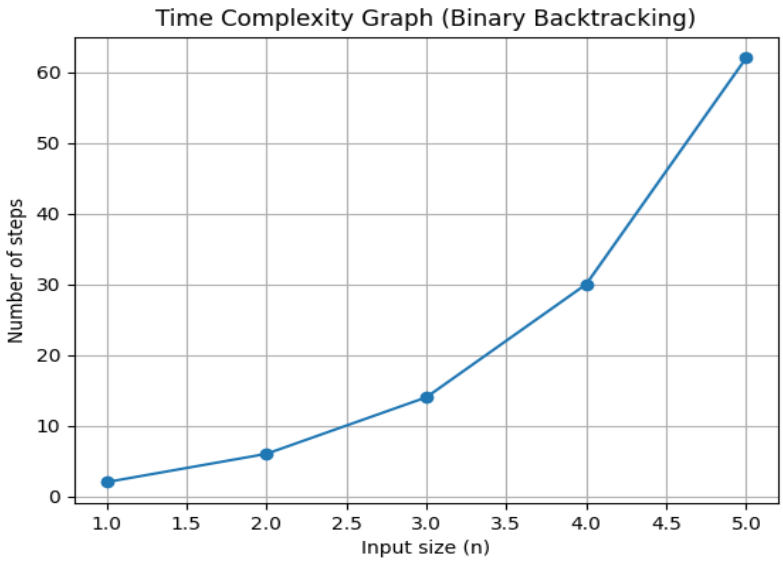
```
binary 3
```

6. Sample Output

```
+ Choose 0
  + Choose 0
    => Output [0 0]
  - Backtrack 0
  + Choose 1
    => Output [0 1]
  - Backtrack 1
- Backtrack 0
```

7. Time Complexity Analysis

Time complexity is demonstrated by counting the number of recursive steps for increasing input sizes. The graph below shows exponential growth for binary backtracking.


Time Complexity Graph (Binary Backtracking)
Number of steps
Input size (n)
0
10
20
30
40
50
60
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

Slide Summary (For Presentation)

Slide 1: Problem

Recursion is hard to visualize because stack behavior is hidden.

Slide 2: Solution

Use C++ for algorithms and Python to visualize execution flow.

Slide 3: Key Idea

Indentation represents recursion depth (stack size).

Slide 4: Choose Step

Represents pushing a new call onto the stack.

Slide 5: Backtrack Step

Represents popping a call from the stack.

Slide 6: Output Step

Represents a valid solution at maximum depth.

Slide 7: Time Complexity

Growth is visualized using operation counts vs input size.

Slide 8: Learning Outcome

Clear understanding of recursion, stack, and backtracking.

Conclusion

This project provides a clear and intuitive way to understand recursive algorithms by making the call stack visible. It serves as an effective educational and demonstration tool.