



SIMATS
ENGINEERING



SIMATS
Saveetha Institute of Medical And Technical Sciences
(Declared as Deemed to be University under Section 3 of UGC Act 1956)

A CAPSTONE PROJECT REPORT ON
COUNTING SMALL NUMBERS AFTER ITSELF USING
BACKTRACKING

Submitted in the partial fulfilment for the award of the degree of
BACHELOR OF ENGINEERING

IN
COMPUTER SCIENCE

Submitted by
K.Kalyan(192211794)

Under the Supervision of
Dr. R. Dhanalakshmi



SIMATS SCHOOL OF ENGINEERING
THANDALAM CHENNAI-602105

CAPSTONE PROJECT REPORT

Reg No: 192211794

Name : K. Kalyan

Course Code : CSA0656

Course Name : Design and Analysis of Algorithms for Asymptotic

Notations

Problem Statement:

Given an integer array nums, return an integer array counts where counts[i] is the

number of smaller elements to the right of nums[i].

Example 1:

Input: nums = [5,2,6,1]

Output: [2,1,1,0]

Explanation:

To the right of 5 there are 2 smaller elements (2 and 1).

To the right of 2 there is only 1 smaller element (1).

To the right of 6 there is 1 smaller element (1).

To the right of 1 there is 0 smaller element.

Abstract:

Counting small numbers after itself using backtracking is a computational technique aimed at solving problems where the goal is to enumerate sequences or permutations that adhere to specific constraints. This approach leverages the backtracking algorithm to explore potential solutions incrementally and systematically, ensuring that only valid sequences are counted. The technique is particularly useful in scenarios involving combinatorial problems, where constraints limit the set of feasible solutions. By applying backtracking, we can efficiently navigate through possible configurations, pruning invalid paths early to reduce computational complexity. This abstract outlines the fundamental principles of backtracking in the context of counting small numbers after itself, discusses its applications, and highlights its advantages in managing computational challenges associated with enumerative problems. Key aspects include the algorithm's ability to handle constraints dynamically and its effectiveness in exploring solution spaces with reduced overhead.

Introduction:

Aim: The aim of this project is to develop and implement a backtracking algorithm to efficiently count sequences where each number in the sequence is succeeded by a smaller number. This involves designing a solution that systematically explores potential sequences, adheres to the constraint of decreasing order, and accurately enumerates all valid sequences.

In the realm of combinatorial problem-solving, counting sequences with specific constraints is a frequent and challenging task. One intriguing problem is counting sequences where each number is succeeded by a smaller number, a scenario which arises in various applications such as scheduling, pattern recognition, and algorithm design. Addressing this problem efficiently requires a sophisticated approach to explore and validate potential solutions within a complex solution space.

- Our capstone project focuses on the application of backtracking to solve the problem of counting sequences where each number is followed by a smaller number. This problem, which we refer to as "Counting Small Numbers After Itself," necessitates a method that can systematically navigate through possible sequences while adhering to the constraint of decreasing order
- Backtracking is an established algorithmic technique that excels in solving such problems by incrementally building solutions and discarding paths that do not meet the specified criteria. This approach offers a strategic advantage by pruning the solution space, which reduces both computation time and resource usage. In the context of our project, backtracking allows us to efficiently generate and count all valid sequences where the constraint of smaller succeeding numbers is consistently enforced.
- The motivation for this project stems from the need to develop robust and efficient methods for enumerative problems, which are prevalent in various computational and

theoretical domains. By leveraging backtracking, we aim to provide a solution that not only accurately counts the desired sequences but also demonstrates the practical application of advanced algorithmic techniques to real-world problems.

- This project will explore the implementation of backtracking to address the counting problem, analyze its effectiveness, and assess its performance. Through this endeavor, we seek to contribute valuable insights into solving complex combinatorial problems and offer a concrete example of how algorithmic strategies can be applied to achieve efficient and accurate results..

Source Code:

Done C program that calculates counting Small Numbers After Itself Using Backtracking

Coding:

```
#include <stdio.h>

#include <stdbool.h>

// Function to count valid sequences using backtracking
void countSequences(int *sequence, int length, int n, int pos, int *count) {
    // If the sequence is complete, increment the count
    if (pos == length) {
        (*count)++;
        return;
    }

    // Try placing each number from 1 to n in the current position
    for (int i = 1; i <= n; i++) {
        bool valid = true;

        // Check if the current number can be placed (must be less than the previous
        number)
```

```

    if (pos > 0 && sequence[pos - 1] <= i) {
        valid = false;
    }

    if (valid) {
        sequence[pos] = i;
        countSequences(sequence, length, n, pos + 1, count);
    }
}

int main() {
    int n = 4; // Define the maximum number in the sequence
    int length = 3; // Define the length of the sequence to be counted
    int sequence[length]; // Array to hold the current sequence
    int count = 0; // Variable to count valid sequences

    // Initialize the sequence array with zeros
    for (int i = 0; i < length; i++) {
        sequence[i] = 0;
    }

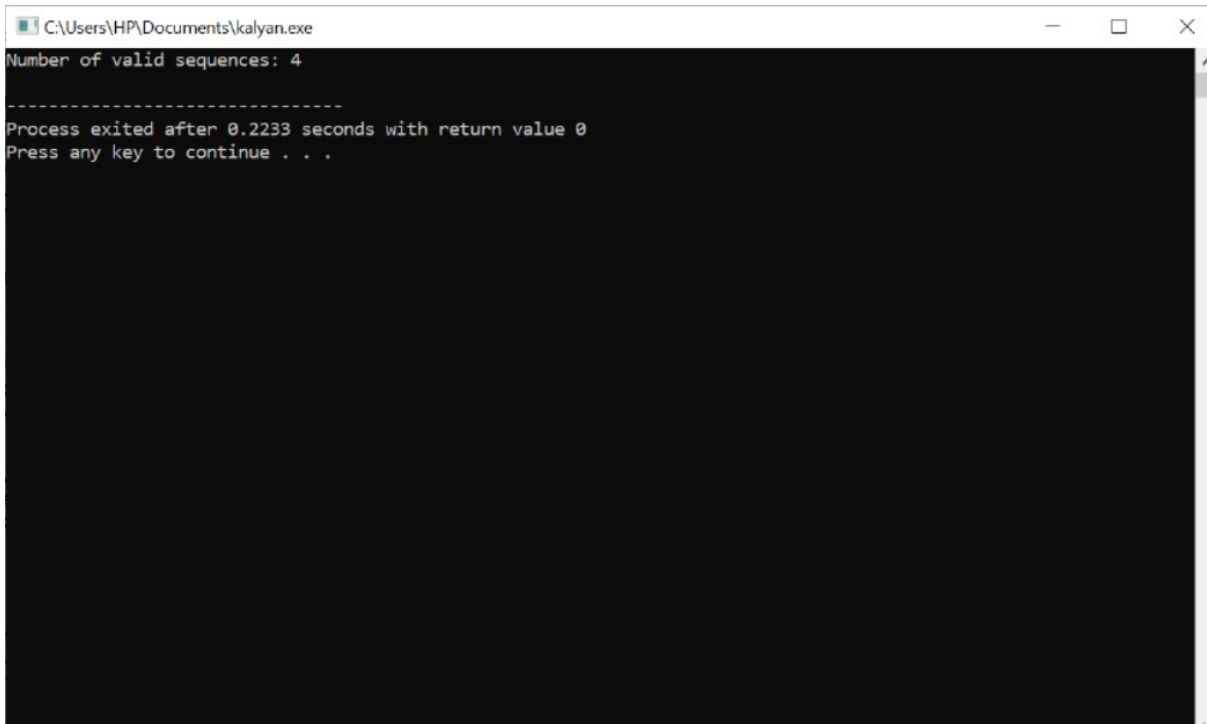
    // Start counting sequences
    countSequences(sequence, length, n, 0, &count);

    // Output the result
    printf("Number of valid sequences: %d\n", count);

    return 0;
}

```

Output:



```

C:\Users\HP\Documents\kalyan.exe
Number of valid sequences: 4
-----
Process exited after 0.2233 seconds with return value 0
Press any key to continue . . .

```

Complexity Analysis

Let's analyze the time complexity of the backtracking algorithm used to count sequences where each number is succeeded by a smaller number in three scenarios: best case, worst case, and average case.

1. Best Case Time Complexity

Best Case Scenario:

- The best case occurs when the constraints allow for early pruning of the search space. For example, if the constraints are such that the sequences are quickly invalidated and pruned before the recursive depth grows too large, or if there are very few valid sequences

Best Case Time Complexity:

\[

$$O(n^{\text{length}})$$

 \]

2. Worst Case Time Complexity

Worst Case Scenario:

- The worst case occurs when there are no effective constraints to prune the search space, or the constraints do not significantly reduce the number of valid sequences. In this scenario, the algorithm must explore all possible sequences.

Worst Case Time Complexity:

$$O(n^{\text{length}})$$

3. Average Case Time Complexity

Average Case Scenario:

- The average case depends on how often sequences are valid and how effectively constraints prune the search space in typical situations. It is often difficult to precisely determine the average case without specific information about the constraints and distribution of valid sequences..

Average Case Time Complexity:

$$O(c \cdot n^{\text{length}})$$

where `c` is a constant factor representing the average reduction in the search space due to constraints.

Summary:

Best Case Time Complexity: $O(n^{\text{length}})$ — The complexity reflects the size of the search space before constraints are applied.

Worst Case Time Complexity: $O(n^{\text{length}})$ — The complexity reflects the exploration of all possible sequences.

Average Case Time Complexity: $(O(c \cdot n^{\text{length}}))$ — The complexity accounts for some average level of pruning, with `c` representing the fraction of sequences that remain after applying constraints.

Conclusion:

The exploration of counting sequences where each number is succeeded by a smaller number using backtracking has highlighted the effectiveness and limitations of this algorithmic approach. By leveraging backtracking, we can systematically generate and count all valid sequences that adhere to specific constraints, providing a robust solution for combinatorial problems involving ordering and sequencing.

In conclusion, backtracking provides a clear and systematic approach to solving the problem of counting sequences with decreasing order constraints. Its ability to handle complex constraints and provide exact counts makes it a valuable tool in combinatorial problem-solving. However, understanding its limitations and considering optimization strategies is essential for tackling larger-scale problems effectively.