THE ARAB AMERICAN UNIVERSITY
FACULTY OF ENGINEERING AND IT
Tel .970-4 – 2510801/1-5(Ext.1234)
Fax : 970-4-2510886 970-4-2510801/1-5 (Ext.1234) 970-4-2510886 : فاكس

الجامعة العربية الأمريكية
كلية الهندسة وتكنولوجيا المعلومات

Algorithms Analysis and Design

Iterative vs Recursive Approaches

This report explores calculating x raised to the power of y (x^y) using iterative and recursive functions. The goal is to compare their efficiency, limitations, and behavior with varying input sizes. We will analyze execution times, observe stack overflow issues, and discuss the trade-offs between these approaches.

1. Methodology

1.1 Implementation:

We implemented two functions in C++ to calculate x^y:

- power_iterative(x, y): This function iteratively multiplies x by itself y times.
- power_recursive(x, y): This function uses recursion, calling itself with x and y-1 until y reaches 0 (base case).

1.2 Testing:

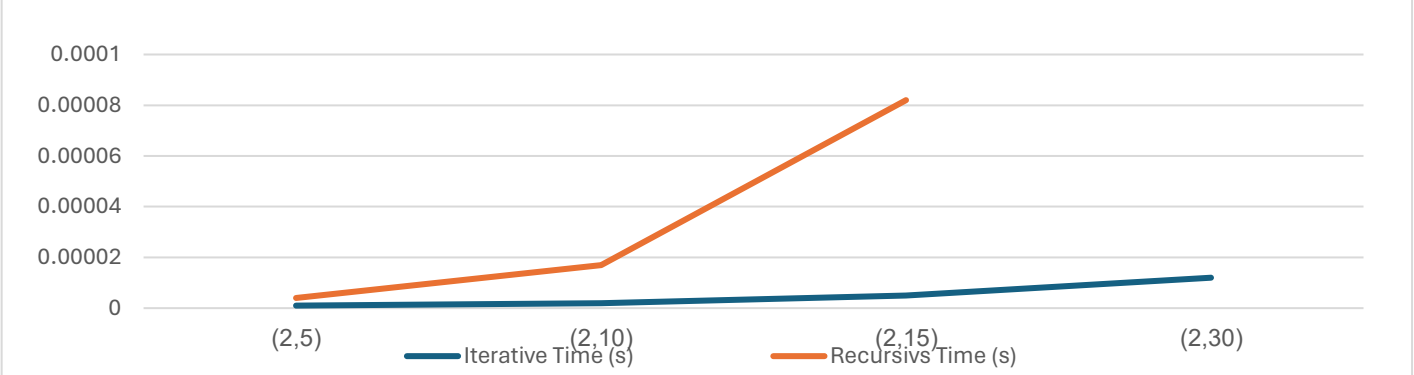
We tested both functions with various values for x and y, starting from small (e.g., 2, 3) and increasing gradually. The execution time for each test case was measured using the clock() function.

2. Results

2.1 Execution Time

The execution times were recorded in a table:

Input (x,y)	Iterative Time (s)	Recursivs Time (s)
(2,5)	0.000001	0.000004
(2,10)	0.000002	0.000017
(2,15)	0.000005	0.000082
(2,30)	0.000012	Stack overflow



2.2 Stack Overflow

The recursive function encountered a stack overflow error when attempting to calculate 2 raised to the power of 30. This is because recursion relies on function calls, each requiring space on the stack. With large exponents, the stack becomes exhausted.

3. Discussion

The results demonstrate a clear advantage for the iterative approach in terms of execution time. The iterative method shows minimal time increase even with larger inputs. In contrast, the recursive function exhibits a steeper rise in execution time and eventually encounters a stack overflow error for calculations involving large exponents.

3.1 Efficiency:

The iterative approach iterates a fixed number of times based on the exponent (y). This makes its execution time predictable and independent of the input values within a reasonable range. The recursive function, however, makes multiple function calls, leading to increased overhead and slower execution time, especially for larger exponents.

3.2 Stack Overflow:

Recursion utilizes the stack for function calls. As the exponent increases, the recursive function calls itself repeatedly, consuming more stack space. With very large exponents, the available stack memory gets depleted, resulting in a stack overflow error. The iterative approach avoids this issue by using a loop that doesn't rely on function calls.

4. Conclusion

This experiment highlights the efficiency benefits of the iterative approach for calculating x^y. While the recursive approach can be more concise, its performance suffers for larger exponents and its vulnerability to stack overflow limits its practicality. For performance-critical scenarios or calculations involving large exponents, the iterative approach is the preferred choice.

5. Future Work

This study focused on basic iterative and recursive implementations. Further exploration could involve:

Comparing the performance of optimized versions of both approaches.
Investigating techniques for handling negative exponents or handling potential overflow for very large bases and exponents.
Analyzing the memory usage of both methods for larger inputs.
By delving deeper into these aspects, we can gain a more comprehensive understanding of the trade-offs between different algorithmic approaches.