Comprehensive Analysis of Fundamental Algorithms in Rust

Thor

November 2, 2025

Contents

Abstract

This report documents the analysis and implementation of core algorithms from "Introduction to Algorithms" (CLRS) using Rust. Focus is on verifying theoretical complexity against practical implementation, starting with Insertion Sort.

Chapter 1

Introduction

1.1 Motivation and Scope

The goal is to bridge theoretical analysis (CLRS) and systems-level Rust programming. Algorithms like Insertion Sort ensure correctness and crate stability.

Chapter 2

Algorithm Analysis: Insertion Sort

2.1 Algorithm Description and Pseudocode

Insertion Sort maintains the sorted subarray invariant.

2.1.1 Pseudocode Listing

```
INSERTION-SORT(A)
  for j = 2 to A.length
    key = A[j]
    i = j - 1
    while i > 0 and A[i] > key
      A[i+1] = A[i]
      i = i - 1
    A[i+1] = key
```

Listing 2.1: Insertion Sort Pseudocode (CLRS)

2.2 Complexity Analysis

Worst-case occurs for reverse-sorted arrays: $T(n) = \Theta(n^2)$.

2.3 Implementation and Verification

2.3.1 Rust Code Listing

```rust
pub fn insertion_sort<T>(arr: &mut [T])
where
    T: PartialOrd + Copy,
{
    if arr.len() < 2 { return; }
    for i in 1..arr.len() {
        let key = arr[i];
        let mut j = i;
        while j > 0 && arr[j - 1] > key {
            arr[j] = arr[j - 1];
            j -= 1;
        }
        arr[j] = key;
    }
}
```

Listing 2.2: Insertion Sort in Rust (`algorithms` Crate)

2.3.2 Execution Example

Running the algorithm with a small, unsorted array $\{5, 2, 4, 6, 1, 3\}$ yields:

```
--- Running Insertion Sort ---
Input: [5, 2, 4, 6, 1, 3]
Output: [1, 2, 3, 4, 5, 6]
```

References