# **Algorithm Analysis Report**

## **Introduction**

Algorithm analysis is a critical aspect of computer science that involves evaluating the efficiency of algorithms in terms of time and space complexity. This report focuses on the implementations of static arrays, dynamic arrays, and string operations, analyzing their performance using Big O notation.

## **Time and Space Complexity**

### **1. Static Array**

#### **Operations:**

* **Insertion**:
  + **Time Complexity**: O(n)
    - In the worst case, insertion requires shifting elements to accommodate the new element.
  + **Space Complexity**: O(1)
    - Only a fixed amount of additional space is used (the array itself).
* **Deletion**:
  + **Time Complexity**: O(n)
    - Similar to insertion, deletion requires shifting elements to fill the gap left by the removed element.
  + **Space Complexity**: O(1)
    - No additional space is used beyond the original array.
* **Traversal**:
  + **Time Complexity**: O(n)
    - Each element must be accessed and printed.
  + **Space Complexity**: O(1)
    - Only a constant amount of space is used for the loop variable.

### **2. Dynamic Array**

#### **Operations:**

* **Insertion**:
  + **Time Complexity**: O(1) on average; O(n) in the worst case (when resizing is needed).
    - Inserting an element at the end is usually O(1), but resizing requires copying elements.
  + **Space Complexity**: O(n)
    - Space grows with the number of elements stored.
* **Deletion**:
  + **Time Complexity**: O(n)
    - Deleting an element requires shifting elements to fill the gap.
  + **Space Complexity**: O(n)
    - Similar to insertion, space grows with the number of elements.
* **Traversal**:
  + **Time Complexity**: O(n)
    - All elements must be accessed.
  + **Space Complexity**: O(1)
    - Only a constant amount of space is used.

### **3. String Operations**

#### **Operations:**

* **Concatenation**:
  + **Time Complexity**: O(m + n)
    - Combining two strings requires accessing all characters from both.
  + **Space Complexity**: O(m + n)
    - A new string is created to hold the combined result.
* **Substring**:
  + **Time Complexity**: O(n)
    - Extracting a substring requires traversing the original string.
  + **Space Complexity**: O(n)
    - A new substring is created, which requires additional space.
* **Comparison**:
  + **Time Complexity**: O(min(m, n))
    - The comparison continues until a mismatch is found or the end of either string is reached.
  + **Space Complexity**: O(1)
    - No additional space is used.
* **Character Frequency**:
  + **Time Complexity**: O(n)
    - Each character is accessed once.
  + **Space Complexity**: O(1)
    - Fixed space for frequency array (size 256 for ASCII characters).

## **Recurrence Relations**

### **Binary Search**

* **Recurrence Relation**: T(n) = T(n/2) + O(1)
* **Solution**:
  + Using the Master Theorem, this resolves to O(log n) for both time complexity.

### **Merge Sort**

* **Recurrence Relation**: T(n) = 2T(n/2) + O(n)
* **Solution**:
  + Applying the Master Theorem, this resolves to O(n log n) for time complexity.

## **Conclusion**

This report highlights the efficiency of static arrays, dynamic arrays, and string operations through their time and space complexities. Understanding these complexities is essential for optimizing algorithms and ensuring effective resource utilization. The recurrence relations for binary search and merge sort illustrate the foundational concepts of algorithm analysis, demonstrating how recursive algorithms can be analyzed and classified.