Algorithms and Data Structures: Module Check-in

Derivation of Time Complexities

Christoph Brauer

28th March 2023

Introduction to Time Complexity

- Measure of efficiency of an algorithm
- Function of the number of input elements
- Helps to compare different algorithms

Importance of Time Complexity in Algorithms

- Optimal resource usage
- Scalability
- Better understanding of the algorithm's behavior

Asymptotic Notations

- Big O Notation (O): Upper bound
- Big Omega Notation (Ω): Lower bound
- Big Theta Notation (Θ): Tight bound

Analyzing Loops (Counting Steps)

- Identify the basic operation
- Count the number of basic operations
- Express the count in terms of input size

Example 1: Linear Algorithm

```python
def linear_algorithm(arr):
    sum = 0
    for x in arr:
        sum += x
    return sum
```

Time Complexity: O(n)

Example 2: Quadratic Algorithm

```python
def quadratic_algorithm(arr):
    count = 0
    for i in range(len(arr)):
        for j in range(i + 1, len(arr)):
            if arr[i] > arr[j]:
                count += 1
    return count
```

Time Complexity: O(n^2)

Example 3: Logarithmic Algorithm

```
def binary_search(arr, target):
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

Time Complexity: O(log n)

Conclusion

- Importance of time complexity in algorithms
- Asymptotic notations: Big O, Omega, and Theta
- Analyzing loops using asymptotic notations