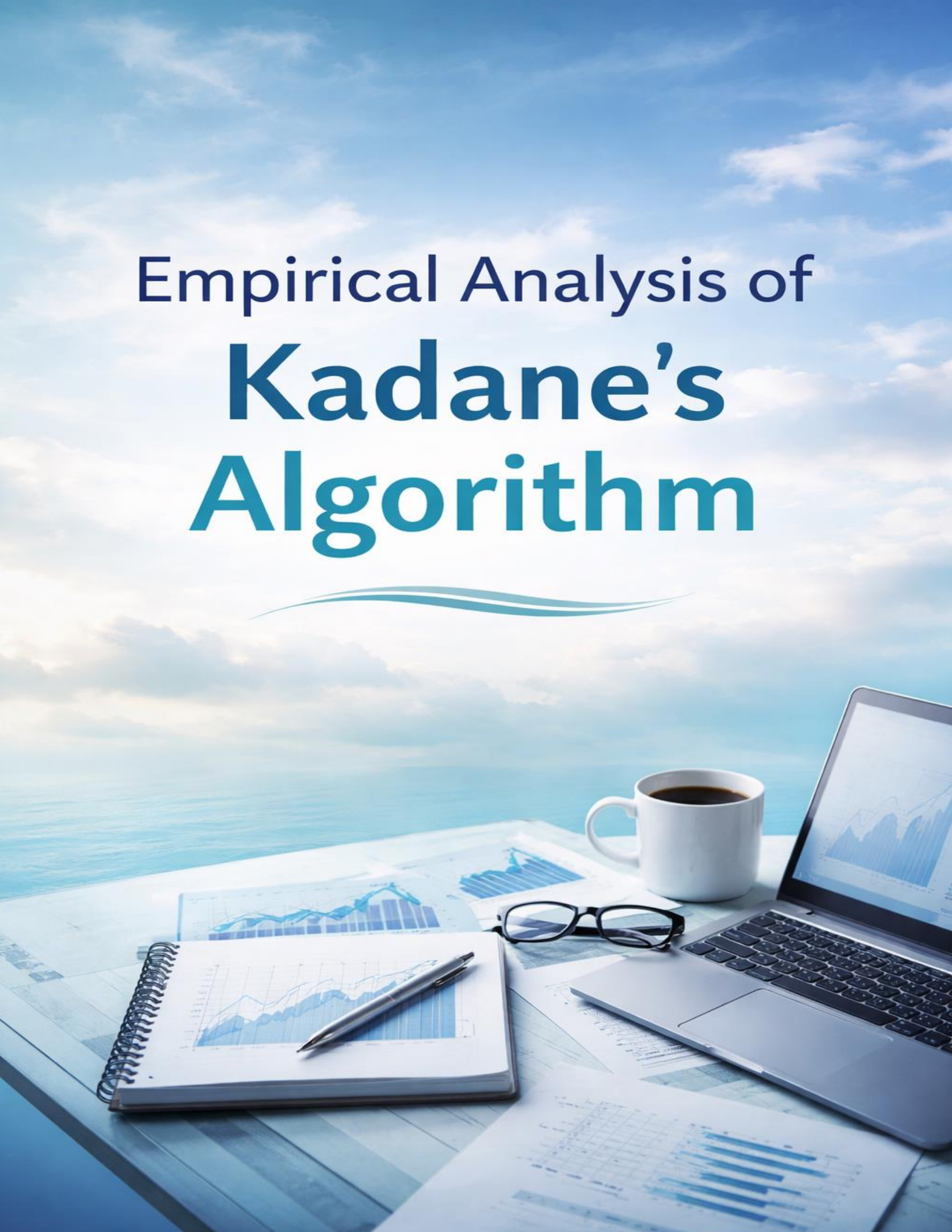
Empirical Analysis of Kadane's Algorithm

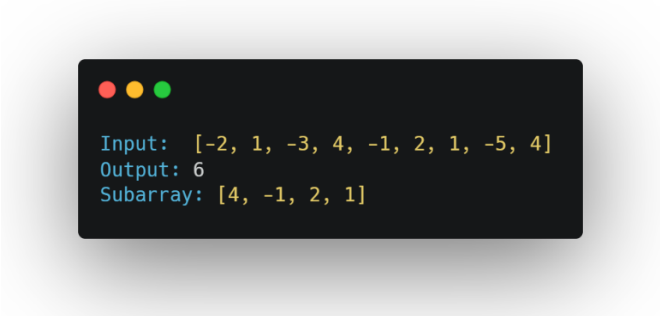


Introduction

- The **Maximum Subarray Sum Problem** is a fundamental problem in computer science that aims to find the contiguous subarray within a one-dimensional array of numbers which has the largest sum. This problem has applications in areas such as data analysis, economics, and signal processing.
- One of the most efficient solutions to this problem is **Kadane's Algorithm**, which provides an optimal linear-time solution.

Problem Definition

Given an array of integers $A = [a_1, a_2, \ldots, a_n]$, the objective is to determine the maximum possible sum of a contiguous subarray.



Algorithm Description

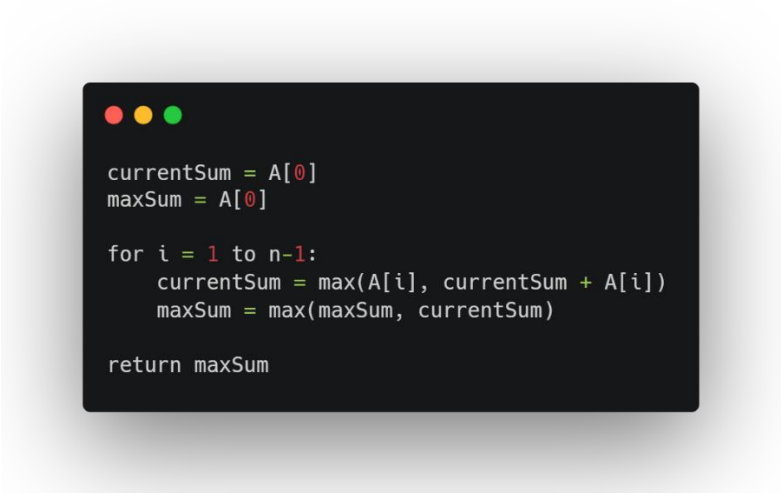
Kadane's Algorithm uses a **dynamic programming approach** where, at each index, one decision is made whether to:

- Extend the previous subarray
- Start a new subarray from the current element

Key Variables:

- currentSum: Maximum sum ending at the current index
- maxSum: Global maximum subarray sum found so far

Pseudocode:



```
currentSum = A[0]
maxSum = A[0]

for i = 1 to n-1:
    currentSum = max(A[i], currentSum + A[i])
    maxSum = max(maxSum, currentSum)

return maxSum
```

Theoretical Analysis

1- Time Complexity

- The algorithm scans the array **once**
- **Time Complexity: O(n)**

2- Space Complexity

- Uses a constant number of variables
- **Space Complexity: O(1)**

Empirical Analysis Methodology

1- Experimental Setup

- Independent Variable: Input size (n)
- Dependent Variable: Execution time (milliseconds)
- Platform: Same hardware and software environment
- Input Data: Randomly generated integer arrays
- Runs: Each experiment was executed multiple times and averaged

2- Collected Empirical Data

Input Size (n)	Execution Time (ms) In Kadane	Execution Time (ms) In Brute Force
1,000	0.05	1000
10,000	0.30	10,000
100,000	3.10	100,000
1,000,000	30.40	1,000,000

3- Observations

- Execution time increases proportionally with input size
- The relationship between input size and time is approximately linear
- Minor variations are due to system-level factors such as CPU scheduling

Empirical vs Theoretical Comparison

Aspect	**Theoretical**	**Empirical**
Time Complexity	O(n)	Linear growth
Space Complexity	O(1)	Constant
Scalability	Efficient	Confirmed
Performance	Optimal	Validated

Edge Case Analysis

- **All Negative Values**: Algorithm correctly returns the element
- **Single Element Array**: Returns that element
- **All Positive Values**: Returns sum of the entire array

Limitations

- Performance depends on hardware and compiler optimization
- Results may slightly vary across different runtime environments
- Only contiguous subarrays are considered