Theoretical Analysis

Maximum Subarray Sum (Kadane's Algorithm):-

1 .Problem Definition

The Maximum Subarray Sum problem aims to find the largest possible sum of a contiguous subarray within a given array of integers that may contain both positive and negative values.

Input:

An array , where

Output:

The maximum possible sum of any contiguous subarray of.

Formally, the problem is defined as finding indices and such that:

\text{Maximize } S(i, j) = \sum_{k=i}^{j} a_k \quad \text{where } 0 \leq i \leq j < n

2 .Naive Approach (Brute Force)

Idea

The simplest approach is to consider all possible subarrays, compute their sums, and select the maximum one.

Algorithm Description

Iterate over all possible starting indices.

For each , iterate over all possible ending indices.

Compute the sum of elements from to.

Keep track of the maximum sum found.

Time Complexity

There are possible subarrays.

If the sum is recomputed for each subarray, the total time complexity becomes:

O(n^3)

Using prefix sums, this can be optimized to:

O(n^2)

Space Complexity

O(1)

Limitations

This approach is inefficient for large input sizes and does not scale well due to its quadratic or cubic time complexity.

3 .Optimized Approach — Kadane’s Algorithm

Idea

Kadane’s Algorithm is based on dynamic programming.

At each position, the algorithm decides whether to:

Extend the existing subarray, or

Start a new subarray at the current element.

This decision is made by comparing the current element with the sum obtained by extending the previous subarray.

4 .Algorithm Explanation

Two variables are maintained:

currentSum: maximum sum of a subarray ending at the current index.

bestSum: maximum sum found so far.

At each index:

\text{currentSum} = \max(a_i, \text{currentSum} + a_i)

\text{bestSum} = \max(\text{bestSum}, \text{currentSum})

5 .Correctness Argument

Kadane’s Algorithm works because:

Any subarray with a negative sum will reduce the total sum of a future subarray.

Therefore, if the accumulated sum becomes negative, it is optimal to discard it and start a new subarray.

The algorithm ensures that at each step, currentSum represents the best possible subarray ending at that index.

Thus, by scanning the array once, the algorithm guarantees finding the maximum subarray sum.

6 .Time and Space Complexity

Metric	**Complexity**
Time Complexity	O(n)
Space Complexity	O(1)

Kadane's Algorithm is optimal for this problem since every element must be examined at least once.

7 .Comparison of Approaches

Approach	**Time Complexity**	**Space Complexity**
Naive (Brute Force)	$O(n^3)$	O(1)
Naive + Prefix Sum	$O(n^2)$	O(n)
Kadane's Algorithm	O(n)	O(1)

8 .Conclusion

The Maximum Subarray Sum problem demonstrates the importance of algorithm optimization.

While the naive approach is simple, it is computationally expensive. Kadane's Algorithm provides an elegant and efficient solution using dynamic programming principles, achieving linear time complexity and constant space usage, making it suitable for large-scale applications.