Maximum Subarray Problem

- Using Kadane's Algorithm

Problem Statement

- Given an array of integers (positive, negative, or zero), find the contiguous subarray with the maximum possible sum.

- Example:
- Array: [-2, 1, -3, 4, -1, 2, 1, -5, 4]
- Output: 6
- Subarray: [4, -1, 2, 1]

Recursive Definition

- Let F(i) be the maximum subarray sum ending at index i.

- Recursive formula:
- $F(i) = \max(A[i], A[i] + F(i-1))$

- Either start new subarray or extend previous.

Recursive Solution

- F(i):
- if i == 0 → return A[0]
- else → return max(A[i], A[i] + F(i-1))

- Final Answer = max(F(i)) for all i

Overlapping Subproblems

- To compute F(i) we compute F(i-1), F(i-2)… repeatedly.
- Same values are recomputed → overlapping subproblems.

DP Insight

- F(i) depends only on F(i-1).
- So we store only previous value.
- This leads to Kadane’s Algorithm.

Kadane’s Idea

- Maintain:
- current_sum → max ending here
- max_sum → global max

- current_sum = max(A[i], current_sum + A[i])
- max_sum = max(max_sum, current_sum)

Detailed Example — Step 1

- Array: [-2, 1, -3, 4, -1, 2, 1, -5, 4]

- i=0 → current_sum = -2, max_sum = -2
- i=1 → max(1, -2+1)=1 → max_sum=1
- i=2 → max(-3, 1-3)=-2 → max_sum=1

Detailed Example — Step 2

- Continuing…

- i=3 → max(4, -2+4)=4 → max_sum=4
- i=4 → max(-1, 4-1)=3 → max_sum=4
- i=5 → max(2, 3+2)=5 → max_sum=5

Detailed Example — Step 3

- Continuing…

- i=6 → max(1, 5+1)=6 → max_sum=6
- i=7 → max(-5, 6-5)=1 → max_sum=6
- i=8 → max(4, 1+4)=5 → max_sum=6

- Final Maximum Sum = 6

Maximum Subarray Identified

- Subarray producing max sum:
- [4, -1, 2, 1]

- Sum = 6

Time Complexity

- Kadane scans array once.

- Time = O(n)
- Better than recursion or brute force.

Space Complexity

- Uses only two variables.

- Space = O(1)

Summary

- Recursive relation → Overlapping subproblems → DP insight → Kadane optimization.

- Time: O(n)
- Space: O(1)

Thank You

- Questions?