Kadane's Algorithm: A 500-Word Report with Use Case and Applications

Kadane’s Algorithm is a dynamic programming approach used to find the maximum sum of a contiguous subarray in an array of integers. It is efficient, elegant, and commonly applied in a variety of problems related to subarray optimization. The primary strength of Kadane’s Algorithm lies in its linear time complexity (O(n)) and constant space usage (O(1)), which makes it ideal for large datasets.

### Core Concept

The idea behind Kadane’s Algorithm is to iterate through the array while keeping track of two variables:

1. \*\*currentMax\*\*: The maximum subarray sum ending at the current index.  
2. \*\*maxSoFar\*\*: The maximum subarray sum encountered so far.

At each step, we evaluate whether extending the current subarray with the current element is more beneficial than starting a new subarray from the current element. The recurrence relation is:

```  
currentMax = max(nums[i], currentMax + nums[i])  
maxSoFar = max(maxSoFar, currentMax)  
```

### Use Case: Maximum Subarray (LeetCode #53)

This classic problem asks us to find the contiguous subarray within an integer array that has the largest sum.

\*\*Example Input:\*\* `[-2,1,-3,4,-1,2,1,-5,4]`   
\*\*Expected Output:\*\* `6`

\*\*Explanation:\*\* The subarray `[4, -1, 2, 1]` gives the maximum sum of 6. Kadane’s Algorithm solves this efficiently in one pass through the array.

### Other LeetCode Problems Using Kadane’s Algorithm

Several other LeetCode problems use variations of Kadane’s Algorithm, often with slight modifications depending on the problem constraints:

- \*\*#53 – Maximum Subarray\*\*: The standard application of Kadane’s Algorithm.  
- \*\*#918 – Maximum Sum Circular Subarray\*\*: Modify Kadane’s to account for circular arrays by combining max and min subarray logic.  
- \*\*#487 – Max Consecutive Ones II\*\*: A variation using sliding windows but can be thought of as a derivative.  
- \*\*#1749 – Maximum Absolute Sum of Any Subarray\*\*: Requires both max and min subarrays to get the maximum absolute difference.  
- \*\*#152 – Maximum Product Subarray\*\*: Similar to Kadane’s, but tracks both maximum and minimum products due to negative numbers.  
- \*\*#1191 – K-Concatenation Maximum Sum\*\*: Involves running Kadane’s on the concatenated array.  
- \*\*#363 – Max Sum of Rectangle No Larger Than K\*\*: Uses Kadane’s in a 2D context.

### Conclusion

Kadane’s Algorithm is not only efficient but also intuitive once understood. It leverages dynamic programming principles, making decisions at each step by considering both current value and previous results. Its adaptability across a wide range of problems on LeetCode demonstrates its utility and relevance in solving real-world optimization challenges.