

3.
1
3

Maximum Subarray (Kadane's)

MEDIUM

■ What You Will Learn

- ✓ Kadane's Algorithm — the most elegant DP trick
- ✓ When to restart a subarray vs extend it
- ✓ $O(n)$ time and $O(1)$ space — one of the most famous DSA algorithms

■ Intuition First

You're a trader who makes or loses money each day. Should you keep the running total or start fresh? Kadane's insight: if your running total goes negative, drop it — starting fresh is always better because a negative prefix only drags the next element down.

■ Kadane's Core Rule

- $\text{currentSum} = \text{max running sum ending at current index}$
- If $\text{currentSum} < 0$: reset $\text{currentSum} = \text{arr}[i]$ (start fresh)
- Else: $\text{currentSum} += \text{arr}[i]$ (extend subarray)
- $\text{maxSum} = \text{max of all currentSum values seen}$

■ Visual Trace on [-2,1,-3,4,-1,2,1,-5,4]

Max subarray = [4,-1,2,1] = 6

-2	1	-3	4	-1	2	1	-5	4
----	---	----	---	----	---	---	----	---

```
i=0: cur=-2 <0 → reset to -2. max=-2
i=1: cur=-2+1=-1 <0 → reset to 1. max=1
i=2: cur=1-3=-2 <0 → reset to -3. max=1
i=3: cur=-3+4=1 >0 → cur=4 (reset since -3 dragged down). max=4
i=4: cur=4-1=3. max=4
i=5: cur=3+2=5. max=5
i=6: cur=5+1=6. max=6
i=7: cur=6-5=1. max=6
i=8: cur=1+4=5. max=6 ✓
```

■ Java Code

```
public static int maxSubarraySum(int[] arr) {
```

```
int currentSum = arr[0];
int maxSum = arr[0];
for (int i = 1; i < arr.length; i++) {
    // Should we extend or start fresh?
    if (currentSum < 0) {
        currentSum = arr[i]; // Fresh start – negative prefix is a burden
    } else {
        currentSum += arr[i]; // Extend the subarray
    }
    maxSum = Math.max(maxSum, currentSum);
}
return maxSum;
}
```

■ Complexity

Case	Time	Space	What happens?
All Cases	O(n)	O(1)	Single pass — one of the fastest possible

■ Common Mistakes

- Initializing maxSum=0 breaks all-negative arrays like [-3,-1,-2] (answer is -1, not 0)
- Always init maxSum = arr[0], not 0
- If problem asks for subarray indices too — track start and end pointers

■ Pro Tips

- Can also write as: `currentSum = max(arr[i], currentSum + arr[i])`
- This is dynamic programming — 'should I extend or start fresh?'
- Extension: maximum circular subarray sum uses this + the inverse

■ Practice Problems

[Med] Print the actual subarray (not just the sum)

[Med] Maximum product subarray

[Hard] Maximum sum circular subarray

■ Quick Revision

- `currentSum` = running sum ending here
- If `currentSum < 0`: reset to `arr[i]`
- `maxSum` = max of all `currentSums`
- Init both with `arr[0]` — not 0
- $O(n)$ time, $O(1)$ space — classic DP