Maximum Subarray Sum - 3 Approaches

1. Brute Force ($O(n^3)$)

Idea: Try all possible subarrays and calculate their sums.

Steps:

- Loop over all starting and ending indices.
- Compute the sum for each subarray.
- Keep track of the maximum sum.

Code:

```
int maxSubarraySum(vector<int>& arr) {
    int n = arr.size(), maxSum = INT_MIN;
    for (int i = 0; i < n; i++)
        for (int j = i; j < n; j++) {
            int sum = 0;
            for (int k = i; k <= j; k++)
                sum += arr[k];
            maxSum = max(maxSum, sum);
        }
    return maxSum;
}
```

2. Better Brute Force using Prefix Sum ($O(n^2)$)

Idea: Precompute prefix sums to reduce time of summing subarrays.

Steps:

- Create a prefix sum array: prefix[i] = sum of arr[0...i]

- Use it to compute subarray sums in O(1) time.

Code:

```
int maxSubarraySum(vector<int>& arr) {
    int n = arr.size();
    vector<int> prefix(n);
    prefix[0] = arr[0];
    for (int i = 1; i < n; i++)
        prefix[i] = prefix[i-1] + arr[i];

    int maxSum = INT_MIN;
    for (int i = 0; i < n; i++)
        for (int j = i; j < n; j++) {
            int sum = prefix[j] - (i > 0 ? prefix[i-1] : 0);
            maxSum = max(maxSum, sum);
        }
    return maxSum;
}
```

3. Kadane's Algorithm (O(n)) [Optimal]

Idea: Use a running sum, reset it when it goes negative.

Steps:

- Traverse the array, at each step choose:
  - either start a new subarray from current element, or
  - continue the previous one.

- Track the maximum sum so far.

Code:

```
int maxSubarraySum(vector<int>& arr) {
    int maxSum = arr[0], currSum = arr[0];
    for (int i = 1; i < arr.size(); i++) {
        currSum = max(arr[i], currSum + arr[i]);
        maxSum = max(maxSum, currSum);
    }
    return maxSum;
}
```