Daily Coding Problem #49

Problem

This problem was asked by Amazon.

Given an array of numbers, find the maximum sum of any contiguous subarray of the array.

For example, given the array [34, -50, 42, 14, -5, 86], the maximum sum would be 137, since we would take elements 42, 14, -5, and 86.

Given the array [-5, -1, -8, -9], the maximum sum would be 0, since we would not take any elements.

Do this in O(N) time.

Solution

The brute force approach here would be to iterate over every contiguous subarray and calculate its sum, keeping track of the largest one seen.

```
def max_subarray_sum(arr):
    current_max = 0
    for i in range(len(arr) - 1):
        for j in range(i, len(arr)):
            current_max = max(current_max, sum(arr[i:j]))
    return current_max
```

This would run in O(N^3) time. How can we make this faster?

We can work backwards from our desired solution by iterating over the array and looking at the maximum possible subarray that can be made ending at each index. At each index, either we can include that element in our sum or we exclude it.

each index, either we can inc.ude that e.ement in our sum or we exc.ude it.

We can then keep track of the maximum subarray we've seen so far in a variable `max_so_far`, compute the maximum subarray that includes `x` at each iteration, and choose whichever one is bigger.

```
def max_subarray_sum(arr):
    max_ending_here = max_so_far = 0
    for x in arr:
        max_ending_here = max(x, max_ending_here + x)
        max_so_far = max(max_so_far, max_ending_here)
    return max_so_far
```

This algorithm is known as Kadane's algorithm, and it runs in O(N) time and O(1) space.
