Maximum Subsequence Sum

1 Problem

Input: An array `A[1...n]` of integers. Negative values are allowed.

Output: The maximum sum among all subsequences in the array, or 0 if all values are negative.

2 Algorithm

2.1 Naive Implementation

A naive implementation is to sum every possible subsequence and keep a running maximum. This approach is $O(n^3)$.

```
for i in [1, n]:
    for j in [i, n]:
        for k in [i, j]:
            sum += A[k]
        max = max(max, sum)
return max
```

2.1.1 Runtime

The runtime is upper-bounded by $O(n^3)$, but it is possible to derive the constants. Note the changes of variables to simplify the sums.

$$\begin{aligned}
\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=i}^{j} 1 &= \sum_{i=1}^{n}\sum_{j=i}^{n}(j-i+1) && (1)\\
&= \sum_{i=1}^{n}\sum_{j=1}^{n-i+1} j && (2)\\
&= \frac{1}{2}\sum_{i=1}^{n}[(n-i+1)(n-i+2)] && (3)\\
&= \frac{1}{2}\sum_{i=1}^{n} i(i+1) && (4)\\
&= \frac{1}{2}n(n+1)(n+2) && (5)
\end{aligned}$$

2.2 Improved Naive Implementation

We can eliminate the innermost loop by keeping a running sum within the j-loop. This sum represents the sum of the i-j subarray. This approach runs in $O(n^2)$.

```
for i in [1, n]:
    for j in [i, n]:
        sum += A[j]
        max = max(max, sum)
return max
```

2.3 Linear Algorithm

```
m = s = 0
for i in [1, n]:
    s = max(s + A[i], 0)
    m = max(m, s)
```

The key idea for this version of the algorithm is that in `s` we hold a running sum that gets reset to 0 whenever it becomes negative, which means that it holds the maximum sum of all subarrays that end at index `i`. Then `m` holds the maximum subsequence sum subsequence encountered at the current iteration of the loop.