1 Main Idea

In the “Maximum Sum Contiguous Subsequence” problem, we are given a sequence S of n numbers, and asked to find the contiguous subsequence with the largest sum. Formally, we want to find the subsequence

$$S[i^* \dots j^*]$$

where $i^*, j^* = \arg\max_{i,j} \sum_{k=i}^{j} S[k]$ for $1 \leq i \leq j \leq n$. To solve this, we can apply dynamic programming, since this problem exhibits optimal substructure and overlapping subproblems. In this algorithm, we use an array to store the maximum sum of a contiguous subsequence ending at each position in the sequence. Specifically, we define the following:

$$\begin{aligned} dp[1] &= S[1] \\ dp[j] &= \max\{dp[j-1] + S[j], S[j]\} \quad \text{for } 2 \leq j \leq n \end{aligned}$$

We then iterate through the sequence, updating the maximum sum and optimal start and end positions of the contiguous subsequence as we go. Finally, we return the contiguous subsequence with the largest sum.

2 Pseudocode

Algorithm 1: Maximum Sum Contiguous Subsequence

```
Input: A sequence $S$ of $n$ numbers
Output: A contiguous subsequence of $S$ with the largest sum
$dp = [0] * n$;
// dp is an array of $n$ elements initialized to 0
$dp[1] = S[1]$;
$max = S[1]$;
$i^*, j^*, i = 1$;
for $j = 2$ to $n$ do
    if $dp[j-1] > 0$ then
        $dp[j] = dp[j-1] + S[j]$;
    end
    else
        $dp[j] = S[j]$;
        $i = j$;
    end
    if $dp[j] > max$ then
        $max = dp[j]$;
        $i^*, j^* = i, j$;
    end
end
return $S[i^* \dots j^*]$;
```

3 Proof of Correctness

We will prove the correctness of the algorithm by induction on the length of the sequence processed.

3.1 Base case

For $j = 1$, $dp[1] = S[1]$, which is correct as it's the only element considered so far. $i^* = j^* = i = 1$, and $max = S[1]$, which are all correct for a sequence of length 1.

3.2 Inductive hypothesis

Assume that for some k, $1 \leq k < n$, the algorithm correctly computes:

1. $dp[k]$ as the maximum sum of a contiguous subsequence ending at position k
2. max as the maximum sum of any contiguous subsequence in $S[1 \dots k]$
3. i^* and j^* as the start and end indices of the maximum sum contiguous subsequence in $S[1 \dots k]$
4. i as the start index of the current contiguous subsequence ending at k

3.3 Inductive step

We need to prove that these properties hold for $k + 1$.

1. For $dp[k + 1]$, we have two cases:
   (a) If $dp[k] > 0$, then $dp[k+1] = dp[k]+S[k+1]$. This is correct because extending the previous subsequence yields a larger sum than $S[k+1]$ alone.
   (b) If $dp[k] \leq 0$, then $dp[k+1] = S[k+1]$. This is correct because starting a new subsequence at $k + 1$ yields a larger sum than extending the previous one.

   Therefore, $dp[k+1]$ correctly represents the maximum sum of a contiguous subsequence ending at position $k + 1$.
2. The algorithm updates max if $dp[k + 1] > max$. This ensures that max always holds the maximum sum of any contiguous subsequence in $S[1 \dots k + 1]$.
3. If $dp[k + 1] > max$, the algorithm updates $i^* = i$ and $j^* = k + 1$. This correctly identifies the new maximum sum contiguous subsequence:
   - i^* is set to i, which is the start of the current subsequence ending at $k + 1$

   - j^* is set to $k+1$, as this is where the new maximum sum subsequence ends

4. The algorithm updates $i = k+1$ when $dp[k] \leq 0$. This correctly marks the start of a new potential maximum sum subsequence, as any subsequence including elements before $k+1$ would have a smaller sum than one starting at $k+1$.

By the principle of mathematical induction, the algorithm correctly computes $dp[j]$, max, i^*, j^*, and i for all j, $1 \leq j \leq n$. Therefore, the algorithm correctly returns the contiguous subsequence with the largest sum.

4 Runtime Analysis

Initializing the array dp takes $O(n)$ time, since we need to allocate space for n elements. Initializing the first element of dp and the variables max, i^*, j^*, and i takes $O(1)$ time. The for loop iterates through the sequence S from $j = 2$ to n, which is in $O(n)$ time. Inside the for loop, we perform constant time operations (comparison, addition, and assignment) to update $dp[j]$, i, and max. Thus, the dominating factor in the runtime is the for loop, which gives a total runtime of $O(n)$.