Problem 9.48. Array \([a_1, \ldots, a_n]\) is input to the MaxSubstringSum algorithm below.

MaxSum ← 0;
for \(i = 1\) to \(n\) do
  for \(j = i\) to \(n\) do
    CurSum ← 0;
    for \(k = i\) to \(j\) do
      CurSum ← CurSum + \(a_k\);
      MaxSum ← max(CurSum, MaxSum);
  return MaxSum;

(a) Assume every operation (addition, assignment, comparison, max, etc.) takes 1 unit of time.

Show that the running time \(T(n)\) is given by:

\[
T(n) = 2 + \sum_{i=1}^{n} \left[ 2 + \sum_{j=i}^{n} \left( 5 + \sum_{k=i}^{j} 2 \right) \right].
\]

(b) Evaluate the triple nested-sum to show that \(T(n) = 2 + \frac{22}{6} n + \frac{5}{2} n^2 + \frac{1}{3} n^3\).

Problem 9.49. Here is a more efficient algorithm than the one in Problem 9.48.

MaxSum ← 0;
for \(i = 1\) to \(n\) do
  CurSum ← 0;
  for \(j = i\) to \(n\) do
    CurSum ← CurSum + \(a_j\);
    MaxSum ← max(CurSum, MaxSum);
return MaxSum;

It helps to implement and test this and the brute-force algorithms on some sample sequences.

(a) Show that the running time is given by: \(T(n) = 2 + \sum_{i=1}^{n} \left( 3 + \sum_{j=i}^{n} 6 \right)\).

(b) Evaluate the double nested-sum to show that \(T(n) = 2 + 6n + 3n^2\).

Problem 9.50. Here is an algorithm for MaxSubstringSum based on recursion.

function \(S(\ell, r)\) return max(0, \(a_\ell\));
mid ← \(\lfloor (\ell + r)/2 \rfloor\);
LMax ← \(S(\ell, \text{mid})\), RMax ← \(S(\text{mid} + 1, r)\);
MidL ← \(a_{\text{mid}}\), MidLMax ← \(a_{\text{mid}}\);
for \(i = \text{mid} - 1\) to \(\ell\) do
  MidL ← MidL + \(a_i\);
  MidLMax ← max(MidLMax, MidL);
MidR ← \(a_{\text{mid} + 1}\), MidRMax ← \(a_{\text{mid} + 1}\);
for \(i = \text{mid} + 2\) to \(r\) do
  MidR ← MidR + \(a_i\);
  MidRMax ← max(MidRMax, MidR);
return max(LMax, RMax, MidLMax);

Let \(T(n)\) be the running time on a sequence of size \(n\). Show that \(T(1) = 3\).

(a) If \(n\) is even, show that \(T(n) = 2T(\frac{n}{2}) + 21 + \sum_{i=1}^{n/2-1} 6 + \sum_{i=n/2+1}^{n} 6 = 2T(\frac{n}{2}) + 6n + 9\).

(b) If \(n\) is odd, show that \(T(n) = T(\frac{n}{2}(n+1)) + T(\frac{n}{2}(n-1)) + 6n + 9\).

(c) Tinker and compute \(T(n)\) for \(n = 1, 2, 3, 4, \ldots\), to verify

<table>
<thead>
<tr>
<th>(n)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T(n))</td>
<td>3</td>
<td>27</td>
<td>57</td>
<td>87</td>
<td>123</td>
<td>159</td>
<td>195</td>
<td>231</td>
<td>273</td>
<td>315</td>
</tr>
</tbody>
</table>

(d) Use induction to prove that \(T(2^n) = (6n + 12) \cdot 2^n - 9\).
(e) Argue by monotonicity that \(T(2^{\log_2 n}) \leq T(n) \leq T(2^{\log_2 n + 1})\) and prove that

\[3n(\log_2 n + 1) - 9 \leq T(n) \leq 12n(\log_2 n + 3) - 9.\]

Compare the bounds with your table in part (c).
Problem 9.51. Here is a very efficient MaxSubstringSum algorithm.

```
CumSum ← 0;
CumMin ← 0;
MaxSum ← 0;
for i = 1 to n do
    CumSum ← CumSum + a_i;
    CumMin ← min(CumSum, CumMin);
    CurMax ← CumSum − CumMin;
    MaxSum ← max(MaxSum, CurMax);
return MaxSum;
```

The algorithm computes the cumulative sum CumSum from i = 1 to n. The insight is that the max-substring-sum over substrings ending at i is CumSum(i) minus the minimum cumulative sum up to i, CumMin (CumMin starts at 0, the sum of the empty sequence). The algorithm maintains CumSum, CumMin and the maximum of CumSum − CumMin.

(a) Show that the running time is given by: \( T(n) = 5 + \sum_{i=1}^{n} 10 \).
(b) Show that \( T(n) = 5 + 10n \).