1 A ***basic arithmetic expression*** is composed of characters from the set $\{1, +, \times\}$ and parentheses. Almost every integer can be represented by more than one basic arithmetic expression. For example, all of the following basic arithmetic expression represent the integer 14:

$$1+1+1+1+1+1+1+1+1+1+1+1+1+1$$
$$((1+1)\times(1+1+1+1+1))+((1+1)\times(1+1))$$
$$(1+1)\times(1+1+1+1+1+1+1)$$
$$(1+1)\times(((1+1+1)\times(1+1))+1)$$

Describe and analyze an algorithm to compute, given an integer n as input, the minimum number of 1's in a basic arithmetic expression whose value is equal to n. The number of parentheses doesn't matter, just the number of 1's. For example, when $n = 14$, your algorithm should return 8, for the final expression above. The running time of your algorithm should be bounded by a small polynomial function of n.

Solution: Let $Min1s(n)$ denote the minimum number of 1s in a basic arithmetic expression with value n. This function obeys the following recurrence:

$$Min1s(n) = \begin{cases} 1 & \text{if } n = 1 \\ \min\left\{ \begin{array}{l} \min\left\{ Min1s(m) + Min1s(n-m) \;\middle|\; 1 \le m \le n/2 \right\} \\ \min\left\{ Min1s(m) + Min1s(n/m) \;\middle|\; \begin{array}{l} 1 \le m \le \sqrt{n} \text{ and} \\ n/m \text{ is an integer} \end{array} \right\} \end{array} \right\} & \text{otherwise} \end{cases}$$

Here are the first twenty values of this function:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	2	3	4	5	5	6	6	6	7	8	7	8	8	8	8	9	8	9	9

We can memoize this function into a one-dimensional array $Min1s[1 \,..\, n]$. Each entry $Min1s[i]$ depends on all entries $Min1s[j]$ with $j < i$, so we can fill the array in increasing index order.

```
MinOnes(n):
  Min1s[1] ← 1
  for i ← 2 to n
      Min1s[i] ← i      // Easy upper bound
      for m ← 1 to i/2
          Min1s[i] ← min{Min1s[i], Min1s[m] + Min1s[i − m]}
          if ⌊i/m⌋ · m = i
              Min1s[i] ← min{Min1s[i], Min1s[m] + Min1s[i/m]}
  return Min1s[n]
```

The resulting algorithm runs in $\boldsymbol{O(n^2)}$ ***time***.

To think about later:

2 Suppose you are given a sequence of integers separated by $+$ and $-$ signs; for example:

$$1+3-2-5+1-6+7$$

You can change the value of this expression by adding parentheses in different places. For example:

$$1 + 3 - 2 - 5 + 1 - 6 + 7 = -1$$
$$(1 + 3 - (2 - 5)) + (1 - 6) + 7 = 9$$
$$(1 + (3 - 2)) - (5 + 1) - (6 + 7) = -17$$

Describe and analyze an algorithm to compute, given a list of integers separated by $+$ and $-$ signs, the maximum possible value the expression can take by adding parentheses. Parentheses must be used only to group additions and subtractions; in particular, do not use them to create implicit multiplication as in $1 + 3(-2)(-5) + 1 - 6 + 7 = 33$.

Solution: Suppose the input consists of an array $X[0 \,..\, 2n]$, where $X[i]$ is an integer for every even index i and $X[i] \in \{+, -\}$ for every odd index i.

Let $Max(i, k)$ and $Min(i, k)$ respectively denote the maximum and minimum values obtainable by parenthesizing the subexpression $X[2i \,..\, 2k]$. We need to compute $Max(0, n)$. These functions obey the following mutual recurrences:

$$Max(i,k) = \begin{cases} X[2i] & \text{if } i = k \\ \max\left\{ \max\left\{ Max(i,j) + Max(j+1,k) \;\middle|\; \begin{array}{c} i \le j < k \\ X[2j+1] = + \end{array} \right\} \right. & \\ \left. \quad \max\left\{ Max(i,j) - Min(j+1,k) \;\middle|\; \begin{array}{c} i \le j < k \\ X[2j+1] = - \end{array} \right\} \right\} & \text{otherwise} \end{cases}$$

$$Min(i,k) = \begin{cases} X[2i] & \text{if } i = k \\ \min\left\{ \min\left\{ Min(i,j) + Min(j+1,k) \;\middle|\; \begin{array}{c} i \le j < k \\ X[2j+1] = + \end{array} \right\} \right. & \\ \left. \quad \min\left\{ Min(i,j) - Max(j+1,k) \;\middle|\; \begin{array}{c} i \le j < k \\ X[2j+1] = - \end{array} \right\} \right\} & \text{otherwise} \end{cases}$$

We can memoize each of these functions into a two-dimensional array. Each entry $Mxx[i, k]$ depends on earlier entries in the same row of the same array, and later entries in the same column *in both arrays*. Thus, we can fill both arrays simultaneously, by considering rows from bottom to top in the outer loop, and considering each row from left to right in the inner loop.

The resulting algorithm (shown on the next page) runs in $\boldsymbol{O(n^3)}$ ***time***. See Figure 1 for pseudo-code.

```
MaxValue(X[0 .. 2n]):
  for i ← n down to 0
      Max[i, i] ← X[2i]
      Min[i, i] ← X[2i]
      for k ← i + 1 to n
          localMax ← −∞
          localMin ← ∞
          for j ← i to k − 1
              if X[2j + 1] = +
                  localMax ← max {localMax,  Max[i, j] + Max[j + 1, k]}
                  localMin ← min {localMin,  Min[i, j] + Min[j + 1, k]}
              else
                  localMax ← max {localMax,  Max[i, j] − Min[j + 1, k]}
                  localMin ← min {localMin,  Min[i, j] − Max[j + 1, k]}
          Max[i, k] ← localMax
          Min[i, k] ← localMin
  return Max[0, n]
```

Figure 1: Pseudo-code