

- 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 n . The running time of your algorithm should be polynomial in n .

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

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

Here are the first twenty values of this function; more values can be found at [The Online Encyclopedia of Integer Sequences \(sequence A005245\)](#).

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 $\text{Min1s}[1..n]$. Each entry $\text{Min1s}[i]$ depends on all entries $\text{Min1s}[j]$ with $j < i$, so we can fill the array in increasing index order.

```

MINONES( $n$ ):
   $\text{Min1s}[1] \leftarrow 1$ 
  for  $i \leftarrow 2$  to  $n$ 
     $\text{Min1s}[i] \leftarrow i$       ⟨⟨Easy upper bound⟩⟩
    for  $m \leftarrow 1$  to  $i/2$ 
       $\text{Min1s}[i] \leftarrow \min \{ \text{Min1s}[i], \text{Min1s}[m] + \text{Min1s}[i-m] \}$ 
    if  $\lfloor i/m \rfloor \cdot m = i$ 
       $\text{Min1s}[i] \leftarrow \min \{ \text{Min1s}[i], \text{Min1s}[m] + \text{Min1s}[i/m] \}$ 
  return  $\text{Min1s}[n]$ 

```

Assuming each arithmetic operation takes $O(1)$ time, we can compute each value $\text{Min1s}[i]$ in $O(n)$ time, so the overall running time of our algorithm is $O(n^2)$. ■

Solution (pseudocode): The following algorithm runs in $O(n^2)$ time:

```

MINONES( $n$ ):
   $Min1s[1] \leftarrow 1$ 
  for  $i \leftarrow 2$  to  $n$ 
     $Min1s[i] \leftarrow i$       ⟨⟨Easy upper bound⟩⟩
    for  $m \leftarrow 1$  to  $i/2$ 
       $Min1s[i] \leftarrow \min \{Min1s[i], Min1s[m] + Min1s[i - m]\}$ 
      if  $\lfloor i/m \rfloor \cdot m = i$ 
         $Min1s[i] \leftarrow \min \{Min1s[i], Min1s[m] + Min1s[i/m]\}$ 
  return  $Min1s[n]$ 

```

Here $Min1s[i]$ stores the minimum number of **1**s in a basic arithmetic expression whose value is equal to i . ■

Rubric: As usual, each of these solutions would be worth full credit.

The fastest algorithms known for this problem were published in 2023 by former Illinois CS PhD student Qizheng He, who learned about the problem as a TA for CS 374. One of He's algorithms computes $MINONES(n)$ for all integers $n \leq N$, given the integer N as input, in $O(N \text{ polylog } N)$ time. The other computes the single value $MINONES(n)$ in $O(n^{0.6154})$ time.

Harder problem to think about later:

2. Describe and analyze an algorithm to compute, given a list of integers separated by $+$ and $-$ signs, the largest 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 .

Because we are allowing both addition and subtraction, we actually need to compute *two* recursive functions. 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:

$$Max(i, k) = \begin{cases} X[2i] & \text{if } i = k \\ \max \left\{ \begin{array}{l} \max \left\{ Max(i, j) + Max(j+1, k) \mid \begin{array}{l} i \leq j < k \\ X[2j+1] = + \end{array} \right\} \\ \max \left\{ Max(i, j) - Min(j+1, k) \mid \begin{array}{l} i \leq j < k \\ X[2j+1] = - \end{array} \right\} \end{array} \right\} & \text{otherwise} \end{cases}$$

$$Min(i, k) = \begin{cases} X[2i] & \text{if } i = k \\ \min \left\{ \begin{array}{l} \min \left\{ Min(i, j) + Min(j+1, k) \mid \begin{array}{l} i \leq j < k \\ X[2j+1] = + \end{array} \right\} \\ \min \left\{ Min(i, j) - Max(j+1, k) \mid \begin{array}{l} i \leq j < k \\ X[2j+1] = - \end{array} \right\} \end{array} \right\} & \text{otherwise} \end{cases}$$

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.

We need $O(n)$ time to compute each array entry $Min[i, k]$ or $Max[i, k]$, so the entire algorithm runs in $O(n^3)$ time. ■

Solution (pseudocode): 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 . The following algorithm runs in $O(n^3)$ time:

```

MAXVALUE( $X[0..2n]$ ):
  for  $i \leftarrow n$  down to 0
     $Max[i, i] \leftarrow X[2i]$ 
     $Min[i, i] \leftarrow X[2i]$ 
    for  $k \leftarrow i + 1$  to  $n$ 
       $localMax \leftarrow -\infty$ 
       $localMin \leftarrow \infty$ 
      for  $j \leftarrow i$  to  $k - 1$ 
        if  $X[2j + 1] = +$ 
           $localMax \leftarrow \max\{localMax, Max[i, j] + Max[j + 1, k]\}$ 
           $localMin \leftarrow \min\{localMin, Min[i, j] + Min[j + 1, k]\}$ 
        else
           $localMax \leftarrow \max\{localMax, Max[i, j] - Min[j + 1, k]\}$ 
           $localMin \leftarrow \min\{localMin, Min[i, j] - Max[j + 1, k]\}$ 
       $Max[i, k] \leftarrow localMax$ 
       $Min[i, k] \leftarrow localMin$ 
  return  $Max[0, n]$ 

```

Here $Max[i, k]$ and $Min[i, k]$ respectively store the maximum and minimum values obtainable by parenthesizing the subexpression $X[2i..2k]$. ■

Rubric: As usual, both of these solutions would be worth full credit.