That Collatz Problem

7. The *collatz* function on positive integers is defined recursively as follows:

```
collatz(1) = 0
if (n is even) collatz(n) = 1+collatz(n/2)
else collatz(n) = 1+collatz(3n+1)
```

For example, collatz(14) = 1+collatz(7), while collatz(7) = 1+collatz(22). Suppose you want to print out collatz(n) for all n from 1 to 100. Your printout would start like this:

```
collatz(1) = 0
collatz(2) = 1
collatz(3) = 7
collatz(4) = 2
collatz(5) = 5
.
.
.
```

How would you write the code?

(a) The obvious method uses recursion. For example:

```
int collatz(int n)
 {
  assert(n > 0);
  if(n == 1) return 0;
  else if (n%2) return 1+collatz(3*n+1);
  else return 1+collatz(n/2);
 }

int main
 {
  for(int = 1; n <= 100; n++)
   cout << "collatz(" << n << ") = " << collatz(n) << endl;
  return 0;
 }
```

A lot of duplication takes place if this message is used. According to my calculations, the function is called 3142 times.

(b) Perhaps you would like to use dynamic programming:

```
collatz[1] = 0;
for(int n = 1; n <= 100; n++)
 if(n%1) collatz[n] = 1+collatz[3*n+1];
 else collatz[n] = 1+collatz[n/2];
```

How big is your array? If $n = 99$, then $f(n) = 3 * 99 + 1 = 298$. That means that your array must contain collatz[n] for n much larger than 100. How large?

We'll say that n is "needed" if the computation of collatz[1]...collatz[100] needs the value of collatz[n]. There are 251 needed numbers. The largest needed number is 9232, but we don't need an array with 9232 entries; we only need to compute collatz[n] for the 251 needed numbers. They must be worked in topological order. For example, the subproblem collatz[99] must be worked *after* the subproblem collatz[298]. There are two serious questions here: what are the needed numbers, and how do we find a topological order of those numbers? Those questions are as hard as the original problem.

The following is a topological order which I computed by sorting the needed numbers by the value of collatz[n].

1 2 4 8 16 5 32 10 64 3 20 21 128 6 40 42 256 12 13 80 84 85 24 26 160 170 48 52 53 340 17 96 104 106 113 34 35 208 226 11 68 69 70 75 22 23 136 140 7 44 45 46 280 14 15 88 90 92 93 28 29 30 184 9 56 58 60 61 18 19 112 116 122 36 37 38 224 232 244 72 74 76 77 81 448 488 25 148 149 152 154 976 49 50 51 296 298 304 325 98 99 100 101 592 650 33 196 197 202 1300 65 66 67 394 404 433 130 131 134 808 866 43 262 268 269 1732 86 87 89 538 577 172 178 179 1154 57 59 358 2308 118 119 4616 39 238 9232 78 79 3077 6154 2051 4102 1367 2734 911 1822 3644 7288 2429 4858 1619 3238 1079 2158 719 1438 479 958 319 638 1276 425 850 283 566 1132 377 754 251 502 167 334 668 1336 445 890 1780 593 1186 395 790 263 526 175 350 700 233 466 155 310 103 206 412 137 274 91 182 364 121 728 242 1456 484 485 161 970 322 323 107 646 214 215 71 430 142 143 47 286 94 95 31 188 190 62 63 376 124 125 41 250 82 83 27 166 54 55 110 220 73 146 292 97

You can see that there is no reasonable way you could solve the problem with straight dynamic programming.

(c) The best way to handle the problem is by memoization. Memoization allows you to compute only the collatz values of the 251 needed numbers, in topological order. You don't even need to think about how to order them topologically; memoization does that automatically! My memoization code works the subproblems in this topological order:

1 2 4 8 16 5 10 3 6 20 40 13 26 52 17 34 11 22 7 14 28 9 12 80 160 53 106 35 70 23 46 15 18 44 88 29 58 19 32 64 21 24 38 76 25 92 184 61 122 244 488 976 325 650 1300 433 866 1732 577 1154 2308 4616 9232 3077 6154 2051 4102 1367 2734 911 1822 3644 7288 2429 4858 1619 3238 1079 2158 719 1438 479 958 319 638 1276 425 850 283 566 1132 377 754 251 502 167 334 668 1336 445 890 1780 593 1186 395 790 263 526 175 350 700 233 466 155 310 103 206 412 137 274 91 182 364 121 242 484 161 322 107 214 71 142 47 94 31 62 124 41 82 27 30 50 100 33 36 56 112 37 152 304 101 202 67 134 268 89 178 59 118 39 42 74 148 49 98 196 65 130 43 68 136 45 48 116 232 77 154 51 54 188 376 125 250 83 166 55 86 172 57 60 728 1456 485 970 323 646 215 430 143 286 95 190 63 66 104 208 69 72 110 220 73 128 256 85 170 340 113 226 75 78 404 808 269 538 179 358 119 238 79 81 84 296 592 197 394 131 262 87 90 140 280 93 96 146 292 97 224 448 149 298 99