# Number Theoretic Algorithms

큰 수의 가감승제: 자릿수에 대한 분할 정복으로

May 19, 2020

## 1 큰 수

큰 수는 보통 자연수 $B$에 대해 다음과 같은 형태로 표현합니다.

$$N = \sum_{i=0}^{n} a_i B^i$$

이 수에 대해 다음과 같은 표기 $N = (a_0, a_1, \cdots, a_n)$을 사용하겠습니다. 또한

$$N = (a_0, a_1, \cdots, a_n, 0, \cdots) = (a_0, a_1, \cdots)$$

의 표기도 사용하는데, 이때에는 어떤 $n$이 있어 $a_k = 0 \ \forall k > n$을 만족한다고 가정합니다.[1] 이때 이 수 $n$을 bound라고 하며, $u$개의 수에 대해 $n$이 각 수의 bound가 되는 경우 $n$을 $u$개의 수의 bound로 정의합니다. $B$는 고정하며, 이 수의 base라고 부릅니다.

덧셈과 뺄셈은 보통 문제가 되지 않습니다. 두 수 $N = (a_0, a_1, \cdots)$과 $M = (b_0, b_1, \cdots)$에 대해 $n \pm m = (a_0 \pm b_0, a_1 \pm b_1, \cdots)$가 성립합니다. 따라서 평범하게 계산해도 자릿수의 선형 시간 안에 계산됩니다.

문제가 되는 것은 곱셈인데, 두 수 $N$, $M$의 bound를 $n$이라고 할 때,

$$\begin{aligned} NM &= \left( \sum_{i=0}^{n} a_i B^i \right) \cdot \left( \sum_{j=0}^{n} b_j B^j \right) \\ &= \sum_{k=0}^{2n} \left( \sum_{i=0}^{k} a_i b_{k-i} \right) \cdot B^k \end{aligned}$$

입니다. 따라서, 단순히 계산하면 $\mathcal{O}(n^2)$의 시간이 걸리게 됩니다. 또 나눗셈은 계산하는 방법은 알지만,[2] 식으로 표현할 수는 없습니다.

우리의 일차적인 목표는 곱셈 및 나눗셈의 **제곱미만***subquadratic* 시간 알고리즘을 구상하는 것입니다. 결론부터 말씀드리자면, 곱셈은 $\mathcal{O}(n \log n \log \log n)$ 시간만으로 계산할 수 있습니다. 그러나 이 방법을 설명하기에는 사전 지식이 상당히 많이 필요해서, 필요한 사전지식을

[1] 이 표기법은 "표기법상의 미묘한 문제"를 해소하기 위한 방편입니다.
[2] 초등학교 때 배웠던 그 방법! 이것도 $n^2$에 비례하는 시간이 걸립니다.

모두 논의하기 전까지는 곱셈의 시간을 표현하기 위해 $\mathcal{O}(M(n))$이라는 표현을 사용합니다. 이 표현은 일반적으로 통용되는 표현입니다. 이외에도, 곱셈 및 나눗셈을 사용하는 다양한 알고리즘의 subquadratic 알고리즘을 구상해 볼 것입니다.

## 2 나눗셈

나눗셈의 시간을 줄이는 기본적인 아이디어는 다음과 같습니다:

$$\frac{N}{M} = \frac{N}{B^u}\frac{B^u}{M} \approx \frac{NM^\star}{B^u} \quad \exists M^\star$$

$u$의 크기를 크게 하면 계산 결과가 정확해지고, 어느 시점부터는 정수 나눗셈의 결과가 정확히 나오기 시작할 것입니다. 식을 이렇게 쓴 이상 우리는 $M^\star$를 **근사하는** 방법을 찾습니다; 즉 어떤 정수 $i$가 존재해서 $M = B^i$인 경우는 $M^\star$를 구하기도, 나누기도 대단히 쉬우므로 더 고려하지 않습니다.

큰 $u$에 대해서 $M^\star$를 충분히 빠르게 찾을 수 있으면 됩니다. 그 전에 $u$가 얼마나 커야 할까요? $M^\star = \lceil B^u/M \rceil$이라 하면, 우리가 원하는 부등식은 $NM^\star < \lfloor \frac{N}{M} + 1 \rfloor \cdot B^u$입니다.[3] 반면 우리가 쓸 수 있는 부등식은 $NM^\star < \frac{N}{M} \cdot B^u + N$이므로, 약간 한계가 있어 보입니다... 증명을 위해서는 다음 부등식

$$\frac{N}{M} \le \left\lfloor \frac{N}{M} + 1 \right\rfloor - \frac{1}{M}$$

을 고려해야만 합니다. 이 부등식을 이용하면, $NM < B^u$일 때,

$$\begin{aligned} NM^\star &< \frac{N}{M} \cdot B^u + N \\ &\le \left( \left\lfloor \frac{N}{M} + 1 \right\rfloor - \frac{1}{M} \right) B^u + N \\ &= \left\lfloor \frac{N}{M} + 1 \right\rfloor B^u + \left( N - \frac{B^u}{M} \right) \\ &< \left\lfloor \frac{N}{M} + 1 \right\rfloor B^u \end{aligned}$$

와 같이 우리가 원하는 부등식이 증명됩니다.

이제 다음과 같은 수열을 생각합니다.

$$x_{k+1} = 2x_k - \left\lfloor \frac{Mx_k^2}{B^u} \right\rfloor$$

[3] $\lfloor N/M \rfloor B^u \le NB^u/M \le N\lceil B^u/M \rceil = NM^\star$는 자명하게 성립합니다.

$\varepsilon_k = M^\star - x_k$로 정의하면,

$$\begin{aligned}
\varepsilon_{k+1} &= 2\varepsilon_k - M^\star + \left\lfloor \frac{M\left(M^\star - \varepsilon_k\right)^2}{B^u} \right\rfloor \\
&= 2\varepsilon_k - M^\star + \left\lfloor \frac{MM^{\star 2} - 2MM^\star\varepsilon_k + M\varepsilon_k^2}{B^u} \right\rfloor \\
&\le 2\varepsilon_k - M^\star + \left( \frac{MM^{\star 2} - 2MM^\star\varepsilon_k + M\varepsilon_k^2}{B^u} \right) \\
&= \left(M^\star - 2\varepsilon_k\right)\left( \frac{MM^\star}{B^u} - 1 \right) + \frac{M\varepsilon_k^2}{B^u} \\
&\le \left(M^\star - 2\varepsilon_k\right)\frac{M}{B^u} + \frac{M\varepsilon_k^2}{B^u} \\
&= \left( \frac{MM^\star}{B^u} - 1 + 1 - 2\frac{\varepsilon_k M}{B^u} \right) + \frac{M\varepsilon_k^2}{B^u} \\
&\le 1 + \frac{1}{N} + \frac{1}{N}\cdot\varepsilon_k^2 \\
&\approx N \cdot \left(\frac{\varepsilon_k}{N}\right)^2
\end{aligned}$$

이고, 이 부등식에서 $\varepsilon_0$가 0 이상 $\frac{M^\star}{2}$ 이하였다면 $\varepsilon_k$들은 모두 0 이상임을 쉽게 알 수 있습니다. $M^\star$는 미리 알 수 없지만, 작은 수로부터 나눗셈 결과를 대강 추정하여 $\varepsilon_0$가 충분히 작게 할 수는 있습니다. 이 점화식으로부터 **자릿수를 두 배씩** 늘려 가면서 $M^\star$를 구할 수 있음을 알 수 있습니다. 따라서 나눗셈 시간은 $\mathcal{O}(M(n)\log n)$이 됩니다.

# 3 자릿수에 대한 분할 정복

분할 정복은 제곱 시간을 제곱미만 시간으로 만드는 테크닉으로 잘 알려져 있습니다. 분할 정복이 이토록 강력한 이유는 naïve한 계산에서 놓치던 중복 계산을 각 부분문제의 관점에서 보도록 **강제하여** 체계적으로 보도록 돕기 때문입니다.

큰 수의 계산을 subquadratic으로 만들고 싶을 때 구체적인 함수나 계산 방법이 없는 경우 시도해 볼 수 있는 방법이 분할 정복입니다. 여기서는 크게 세 가지 문제인 곱셈, 진법 변환 그리고 최대공약수 문제를 다루어 보겠습니다.

## 3.1 곱셈

먼저 가장 쉬운 곱셈부터 해 봅시다. 두 수 $N = N_1B^n + N_0$, $M = M_1B^n + M_0$이 주어져 있다면,

$$\begin{aligned}
NM &= (N_1B^n + N_0)(M_1B^n + M_0) \\
&= N_1M_1B^{2n} + (N_1M_0 + N_0M_1)B^n + N_0M_0 \\
&= N_1M_1B^{2n} + ((N_1 + N_0)(M_1 + M_0) - N_1M_1 - N_0M_0)B^n + N_0M_0
\end{aligned}$$

와 같이 계산합니다. 그러면 절반의 자릿수를 세 번 계산하도록 **할 수 있으므로**, Master theorem 등에 의해 $\mathcal{O}(n^{\log_2 3})$ 시간에 곱셈을 시행할 수 있습니다.

여기서 중요하게 봐야 할 것은 $M(n) = \mathcal{O}(n^{\log_2 3})$이라는 사실이 아니라, 놓치고 있던 어느 중복 계산을 통해 시간 복잡도가 줄어들었는지입니다. 이런 최적화를 했다 하더라도, $n = 1$로 두어 한 자리 수에 대해서만 계산했다면 최적화가 일어나지 않았을 것입니다.

이곳에서 중요하게 쓴 사실은 $N_0M_0$, $N_1M_1$, $N_0M_1 + N_1M_0$와 일차종속이 되는 곱셈 한 번의 식을 찾으려고 노력했다는 사실입니다. 그런 식을 찾을 수 있게 된다면, 맨 마지막 식을 직접 계산하는 것은 시간 낭비가 됩니다. 관점을 이렇게 바꾸어 볼 수 있었다는 것은 분할 정복에 내재한 특성 때문입니다: 부분문제를 **명시적으로** 정의해야만 분할 정복을 쓸 수 있다.

### 3.2 진법 변환

진법 변환은, $B$진법의 큰 수를 같은 수를 표현하는 $C$진법로 바꾸는 작업입니다. 일반적으로 $C$로의 나눗셈을 통해 $C$진법 수를 얻어내는 작업은, 예상하셨다시피 나눗셈에서 정보를 효율적으로 얻어내고 있지 못합니다. 위와 마찬가지로, 수를 절반으로 쪼개 진법 변환을 한다고 합시다. $B$진법 수열에서 $C$진법 수열로 가는 변환 함수를 $f$라 할 때, $f(a \cdot B^n + b) = f(a) \times f(B^n) + f(b)$를 계산하면 됩니다. 크기 반인 세 부분문제로 쪼갰고, 곱셈 과정이 $M(n) = \mathcal{O}(n^{\log_2 3})$이라 진법 변환이 $\mathcal{O}(n^{\log_2 3} \log n)$이 됩니다!

만약 수를 $a \cdot C^n + b$ 꼴로 나타낼 수 있으면 $f(a \cdot C^n + b) = f(a) \times f(C^n) + f(b)$이고, $f(C^n)$은 생각할 필요가 없으므로 부분문제 두 개로 쪼갤 수 있습니다! 즉 나눗셈을 시행하면 $a$와 $b$를 구할 수 있습니다. 그러나 이 과정은 $f^{-1}(C^n)$, 즉 $B$진법으로 나타낸 $C^n$을 계산해야 합니다. Repeated Squaring 방법을 이용하면 두 방법 모두 본질적인 문제를 해결해, $\mathcal{O}(M(n) \log n)$ 시간에 진법 변환을 시행할 수 있게 해 줍니다.

### 3.3 최대공약수

최대공약수도 마찬가지 방법으로 시간 복잡도를 줄일 수 있는데, 전보다는 조금 까다롭지만 방법론 자체는 동일합니다. 두 수의 최대공약수를 만들 때 놓치고 있는 정보를 생각해 봅시다. 쉬운 답변은 "중간 과정"인데, 더욱 중요한 것은 각 중간 과정이 어떤 선형 결합인지에 대한 정보입니다:

| $a_n = r_{n-2}$ | $b_n = r_{n-1}$ | $r_n = p_n\mathbf{a} + q_n\mathbf{b}$ | $(p_n, q_n)$ |
|---|---|---|---|
| 1581 | 1394 | 187 | (1, -1) |
| 1394 | 187 | 85 | (-7, 8) |
| 187 | 85 | 17 | (15, -17) |
| 85 | 17 | 0 | (-82, 93) |

이 선형 결합의 계수는 크게 세 가지 의미가 있습니다.

- 어떤 과정을 거쳐 $a$와 $b$의 최대공약수를 구했는지를 알려줍니다. 이 과정은 최대공약수 자체를 구하는 데에는 크게 중요하지 않으나, 예를 들어 $ax + by = k$ 꼴의 방정식을 풀거나 할 때 유용하게 사용될 수 있는 정보입니다.

- 최대공약수를 구할 때 쓴 나눗셈 등 모든 과정을 건너뛸 수 있습니다. (Lehmer)
- $a$와 $b$가 포함된 수의 최대공약수에서 이 부분의 digit을 "충분히 없앤다"는 보장이 되어 있는 계수입니다.[4]

우리는 세 번째 성질에 주목하는데, 이를테면 $A = 15810964$, $B = 13949049$에 대해서,

$$U \cdot \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} 30627 \\ 762509 \end{pmatrix} \quad \text{where} \quad U = \begin{pmatrix} 15 & -17 \\ -82 & 93 \end{pmatrix}$$

입니다. 사실 $\det U = \pm 1$인 어떤 행렬에 대해서도 최대공약수는 같을 것이나, 특히 최대공약수의 계산 결과 계수로 나온 행렬 $U$는 더 큰 수에 대해서 그 수의 윗부분을 확실히 잘라냄을 보여 주고 있습니다.

우리의 목표는 $\gcd(a, b)$를 구하기 위해 $\gcd(a, b) = \gcd(a', b')$인 크기가 반인 두 수 $a'$, $b'$과 변환 행렬 $U$를 주는 함수를 구성하는 것입니다. 이 과정은 **half GCD**라고 부릅니다.

이때 $U$의 entry의 길이와 $a'$, $b'$의 길이가 대강 비슷하게 될 것입니다. 따라서 $a$, $b$의 길이를 $n$이라 하면, 먼저 $a$, $b$의 위 $n/2$자리를 잘라내 half GCD를 시행하여 $(a'_0, b'_0, U)$를 얻고, $a$, $b$에 $U$를 apply하면 남은 수의 길이가 $3n/4$쯤 됩니다.[5] 여기서 다시 위 $n/2$를 잘라내 half GCD를 시행하여 $(a'_1, b'_1, U')$을 얻고, 비슷한 방식으로 남은 수의 길이가 $n/2$이 되도록 할 수 있습니다. 우리가 돌려주어야 할 행렬 $U'U$의 entry 길이는 역시 $n/2$쯤 되는데, 이것은 $U$와 $U'$의 entry 길이가 대강 $n/4$쯤 되기 때문입니다.

이때 $\mathrm{hgcd}(n)$을 길이가 $n$인 수에 대해 half GCD를 시행하는 데 걸리는 시간이라고 하면, $\mathrm{hgcd}(n) = 2\,\mathrm{hgcd}(n/2) + \mathcal{O}(M(n))$이므로 half GCD의 소모 시간은 $\mathcal{O}(M(n) \log n)$입니다. half GCD를 $\log n$번 반복하면 GCD를 구할 수 있고, 그 크기가 매 시행마다 반이 되므로 최대공약수를 $\mathcal{O}(M(n) \log n)$에 구할 수 있습니다!

이렇듯 자릿수에 대한 분할 정복 방법은 시간 복잡도를 줄여야 할 때 가장 먼저 시도해 볼 만한 가치가 있는 테크닉입니다.

## 4 문제

1. (a) 다음을 증명하세요. (Hint: $N$을 $M$으로 나누세요.)

$$\frac{N}{M} \leq \left\lfloor \frac{N}{M} + 1 \right\rfloor - \frac{1}{M}$$

   (b) $0 \leq \varepsilon_0 \leq \frac{M^\star}{2}$라면, 왜 모든 $k$에 대해 $\varepsilon_k \geq 0$입니까? Hint.[6]

[4] 자릿수는 선형적이므로, 각 부분이 (roughly) 독립적으로 작용한다고 생각할 수 있습니다.

[5] 위 예제에서, $-82a + 93b = 0$이어서 leading digit의 반을 없앨 것으로 기대했지만, 실제 apply했을 때는 trailing digit이 차이를 만들어내 수의 크기가 효과적으로 줄지는 않았습니다. 여기서는 trailing digit의 길이가 $n/2$, $U$의 entry의 길이가 $n/4$이므로 계산 결과가 $3n/4$보다 작음이 보장됩니다.

[6] $\varepsilon_k$는 정수입니다; 등식과 부등식으로 이어진 chain에서 확실히 0 이상인 항을 찾고, 그 항과 $\varepsilon_k$가 1 미만의 차이를 가짐을 보이세요.

(c) $\varepsilon_k$와 $\varepsilon_{k+1}$에 대한 위 부등식만으로는 $\varepsilon_k \leq 1$일 때 $\varepsilon_{k+1}$이 1일지 0일지 알 수 없습니다. 사실 $\varepsilon_k \leq 1$일 때 $\varepsilon_{k+1} = 0$임을 증명하세요.

2. ($\mathcal{O}(M(n))$ Newton-Raphson Division) 나눗셈 초반에 수의 하위 자리에 들어가는 정보는, $\varepsilon_k$가 줄어드는 경향을 보면, 쓸데없이 정확하다는 생각이 듭니다. 이 부분을 최적화하면 시간 복잡도를 줄일 수 있지 않을까요?

(a) 어떤 정수 $N_k$와 $e_k$에 대해 $x_k = N_k B^{u-e_k}$라면, $x_{k+1}$을 계산하는 과정은 $\mathcal{O}(m \log m)$입니다. 이제 $M$ 대신

$$M_k = \left\lfloor \frac{M}{B^{u-e_k}} \right\rfloor \cdot B^{u-e_k}$$

을 사용해도, 값을 제대로 산출함을 보이세요. (Hint: $e_k \geq u$일 때까지 "존버.")

(b) $e_k \geq u$인 최소의 $k$에 도달했을 때 $\varepsilon_k = \mathcal{O}(\sqrt{M^\star})$임을 보이세요. 이로써, $k$번 이후로 수렴값을 얻기까지의 iteration이 상수 번입니다.

(c) $i$에 대해 바뀐 점화식을 계산하는 데 $\mathcal{O}(M(e_i))$임을 보이고, $e_{i+1} = 2e_i$이 가능함을 보여 전체 과정이 $\mathcal{O}(M(n))$임을 보이세요.

3. (Toom-3) 왜 항상 두 개로 쪼개는 걸까요? 꼭 둘로 쪼갤 필요는 없습니다: $N = N_2 B^{2n} + N_1 B^n + N_0$, $M = M_2 B^{2n} + M_1 B^n + M_0$가 주어지면, $NM = \sum_{i=0}^{4} X_i B^{in}$일 때,

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{3} & -1 & \frac{1}{6} & -2 \\ -1 & \frac{1}{2} & \frac{1}{2} & 0 & -1 \\ -\frac{1}{2} & \frac{1}{6} & \frac{1}{2} & -\frac{1}{6} & 2 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} M_0 N_0 \\ (M_0 + M_1 + M_2)(N_0 + N_1 + N_2) \\ (M_0 - M_1 + M_2)(N_0 - N_1 + N_2) \\ (M_0 - 2M_1 + 4M_2)(N_0 - 2N_1 + 4N_2) \\ M_2 N_2 \end{pmatrix}$$

임을 확인하세요.[7] 시간 복잡도는 $M(n) = \mathcal{O}(n^{\log_3 5})$입니다.

4. (Repeated Squaring)

(a) 진법 변환에서, $u = 2^k$으로만 두어도 $\mathcal{O}(M(n) \log n)$ 시간 복잡도를 보장함을 보이세요.

(b) $f\left(B^{2^k}\right)$를 필요한 만큼 구하는 데 $\mathcal{O}(M(n))$임을 보이세요. 따라서 진법 변환을 시행하는 데는 $\mathcal{O}(M(n) \log n)$ 시간이 걸립니다. 만족스럽군요.

(c) $f^{-1}\left(C^{2^k}\right)$을 계산하는 데에도 (a), (b)를 그대로 적용할 수 있음을 보이세요.[8]

5. (Major Flaw on Naïve Half GCD and Robust HGCD)

[7] 이걸 대체 어떻게 알았을까요? 일반적으로 이 식을 얻어내는 **방법**을 알게 되면, 그 방법을 자연스럽게 확장해 $\mathcal{O}(n \log n \log\log n)$ 곱셈을 얻을 수 있습니다.

[8] 따라서 어느 쪽이든 시간 복잡도는 $\mathcal{O}(M(n) \log n)$입니다. 그러나 특히 계산이 빠른 진법, 예를 들어 2진법이 아니면 나눗셈이 들어가면 느려지기 때문에 보통 전자, 즉 $B$진법에서 반으로 나눠서 합치는 방법을 씁니다.

(a) $n_0 = 35748972$, $m_0 = 23832648$에 대해 HGCD를 시행하기 위해, 앞 4자리에 대해서 quadratic gcd를 수행했습니다:

| $a_n$ | $b_n$ | $r_n = p_n a_0 + q_n b_0$ | $(p_n, q_n)$ |
|---|---|---|---|
| 3574 | 2383 | 1191 | (1, -1) |
| 2383 | 1191 | 1 | (-2, 3) |
| 1191 | 1 | 0 | (2383, -3574) |

$U = \begin{pmatrix} 1 & -1 \\ -2 & 3 \end{pmatrix}$나 $U = \begin{pmatrix} -2 & 3 \\ 2383 & -3574 \end{pmatrix}$ 모두 두 수의 자릿수를 6 이하로 줄이지 않음을 보이세요. 따라서 HGCD의 대전제가 흔들립니다.

(b) (a)에서의 문제는 HGCD가 실제로 "자릿수를 반으로 줄이는" 결과가 나오지 않을 수도 있다는 점입니다. 상식적으로, $g$가 자릿수의 반보다 크다면, 자릿수의 반이 되는 두 수의 최대공약수가 $g$가 될 수는 없습니다. 그러나 위의 예제처럼 두 수 중 한 개의 수는 자릿수를 반 이하로 줄일 수 있습니다.[9] 이것을 엄밀하게 보이기 위해, 먼저 $|p_n|$과 $|q_n|$이 모두 $\frac{a}{r_{n-1}}$보다 작음을 보이세요. (Hint: $\det U = \pm 1$을 이용하여 $\begin{pmatrix} p_n & r_n \\ p_{n+1} & r_{n+1} \end{pmatrix}$의 determinant가 $a$ 이하임을 보이세요.)

(c) (b)의 결과를 바탕으로 두 수 중 큰 쪽의 원래 자릿수를 $k$라 하면, HGCD 과정이 0을 반환하지 않는 경우에는 자릿수를 $(\lceil k/2 \rceil + 4)$ 이하로 줄임을 보이세요. 따라서 과정을 $\log_2 k$번 시행하면 길이가 $c \log_2 k + \mathcal{O}(1)$ 이하로 떨어지고, 여기서 (super-)quadratic gcd를 사용하면 원래의 시간복잡도가 보장됩니다.

6. (Lehmer's GCD Algorithm)

(a) 임의의 자연수 $N$, $M$에 대해 뒤 $k$자리를 제외하고 $(p_n, q_n)$을 구할 경우, $p_n N + q_n M$이 $(r_n + p_n)B^k$과 $(r_n + q_n)B^k$ 사이에 있음을 보이세요.

(b) 따라서 만일

$$\left\lfloor \frac{a_n + p_n}{b_n + p_{n+1}} \right\rfloor = \left\lfloor \frac{a_n + q_n}{b_n + q_{n+1}} \right\rfloor$$

이면 앞 자리만으로 $p_{n+2}$와 $q_{n+2}$를 유일하게 결정할 수 있음을 보이세요. 따라서 어떤 step의 긴 자리 나눗셈을 없앴습니다.

(c) 이 알고리즘이, $B$가 충분히 크다면, 긴 자리 나눗셈 횟수의 반 이상을 없앰을 보이세요. 시간 복잡도는 여전히 $\mathcal{O}(nM(n))$이고 만일 $M(n) = \mathcal{O}(n \log n \log\log n)$ 이면 제곱 정도의 시간이지만, 긴 자리 나눗셈 횟수가 significantly 줄기 때문에 HGCD의 base case gcd algorithm으로 많이 채택됩니다.

[9] 만일 $g$가 자릿수의 반보다 크다면 그 둘 중 하나의 수가 0이 될 것이고, 아니면 나눗셈을 한 번 시행해 두 수 모두 자릿수를 반 이하로 떨어뜨리면 됩니다.