

# 고속 역제곱근 알고리즘

한남대학교 수학과

20172581 김남훈

## 1. 고속 역제곱근 알고리즘

고속 역 제곱근 알고리즘은 UC버클리 수학과 교수인 윌리엄 카한이 1986년 아이디어를 제시하고, 그 아이디어에 기반하여 실리콘 그래픽스 사가 90년대 초 개발한 알고리즘으로, 간단히 설명하면 실수  $a$  에 대해  $a^{-\frac{1}{2}}$  를 빠르게 계산하는 알고리즘이다. 우선 코드를 보면 다음과 같다.

```
float q_rsqrt(float number)
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y;                                // evil floating point bit level
hacking
    i = 0x5f3759df - ( i >> 1 );                          // what the fuck?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
    // y = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this can be
    removed

    return y;
}
```

출처: [https://en.wikipedia.org/wiki/Fast\\_inverse\\_square\\_root](https://en.wikipedia.org/wiki/Fast_inverse_square_root)

위 코드는 1999년 발매된 게임 **퀘이크 3 아레나** 의 소스코드 중 일부로, 유명 프로그래머인 존 카맥이 작성했다고 알려져 있으며, 주석에는 이해할 수 없는 코드에 대한 동료 프로그래머의 당혹감이 담겨 있다. 위 함수가 어떻게  $(\text{number})^{-\frac{1}{2}}$  을 반환하는지 지금부터 알아보자.

## 2. 컴퓨터가 숫자를 저장하는 방법

컴퓨터에 숫자가 저장되는 방법은 크게 두 가지, 정수형과 실수형으로 나눌 수 있다. 컴퓨터 메모리는 0 과 1 만을 저장할 수 있으므로, 정수형과 실수형 모두 기본적으로 이진법으로 저장된다. 이 때, 0 또는 1 의 값을 갖는 각 자리수를 비트(bit) 라고 한다.

### 정수형

정수형은 주어진 정수를 다음과 같이 32 자리 또는 64 자리의 이진법으로 저장한다. 이 때, 첫 번째 비트는 수의 부호(0 이면 양수, 1 이면 음수) 를 나타내며, 뒤의 비트들은 이진수의 각 자릿수를 나타낸다.

$$537 = 0\ 0000000\ 00000000\ 00000010\ 00011001_{(2)}$$

### 실수형

실수형은 다른 말로 부동소수점(floating point)이라고 한다.

수학과 학생들은 다음과 같은 표현에 익숙할 것이다.

$$373.884 = 3.73884 \times 10^2$$

여기서 3.73884 을 유효숫자, 2 를 지수 라고 한다. 컴퓨터가 실수를 저장하는 방법도 동일하다.

$$\begin{aligned} 373.884 &= 101110101.11100010010011011101_{(2)} \\ &= 1.0111010111100010010011011101_{(2)} * 2^8 \end{aligned}$$

현대의 표준 규격은 부호를 나타내는 1 비트, 지수를 나타내는 8 비트, 유효숫자를 나타내는 23 비트로 총 32 비트로 실수를 저장한다. 표준 규격대로 위의 수를 저장하면

$$373.884 \simeq 0 \underbrace{10000111}_8 \underbrace{01110101111000100100110}_{23}_{(2)}$$

가 된다. 여기서, 지수가  $00001000_{(2)}$  이 아니라  $10000111_{(2)}$  인 이유는, 지수에  $2^8 - 1 = 127$  을 더해 저장하기 때문이다. 이러한 방식의 장점은 1 보다 작은 실수도 저장할 수 있다는 것이다. 실제로 32 비트 부동소수점 방식은  $2^{-127} \sim 2^{127}$  (약  $10^{-38} \sim 10^{38}$ ) 사이의 실수를 저장할 수 있다.

## 2. 정수형과 실수형의 관계

고속 역제곱근 알고리즘에는 실수형 변수를 정수형으로 인식시키는 과정과, 반대로 정수형 데이터를 실수형으로 인식시키는 과정이 존재한다. 이것을 이해하기 위해, 먼저 정수형과 실수형의 관계를 수학적으로 나타내보자.

먼저, 실수형 변수  $x$  를 이루는 32 비트 중 중 맨 앞의 1 비트를  $s$ , 뒤의 8 비트를  $a$ , 그 뒤의 23 비트를  $b$  라 하자.

$$3.14 = 0 \underbrace{10000000}_s \underbrace{10010001111010111000010}_a \underbrace{10010001111010111000010}_b$$

실수형 변수를  $s, a, b$  로 나누는 예시

이제 이것을 정수로 인식시켰을 때 받아들이는 값을  $y$  라 하자. 그러면  $y$  는 다음과 같이 표현된다.

$$y = (-1)^s (a \times 2^{23} + b)$$

## 3. Newton-Raphson Method

수치해석학을 수강한 학생이라면 들어 보았을 Newton-Raphson Method(이하 뉴턴법)는 주어진 방정식의 근사해를 찾는 수치해석적 방법이다.

미분 가능한 함수  $f$  에 대해 방정식

$$f(x) = 0$$

이 주어졌을 때, 뉴턴법은 다음과 같은 절차를 통해 방정식의 근사해  $x$  를 찾는다.

1. 임의의 실수  $x_0$  을 선택한다.
2. 다음 과정을 반복한다.
  1.  $x = x_i$  에서  $f(x)$  의 접선  $l$  을 구한다.
  2.  $l$  의  $x$  절편을  $x_{i+1}$  로 놓는다.

그래프로 보면 다음과 같다.

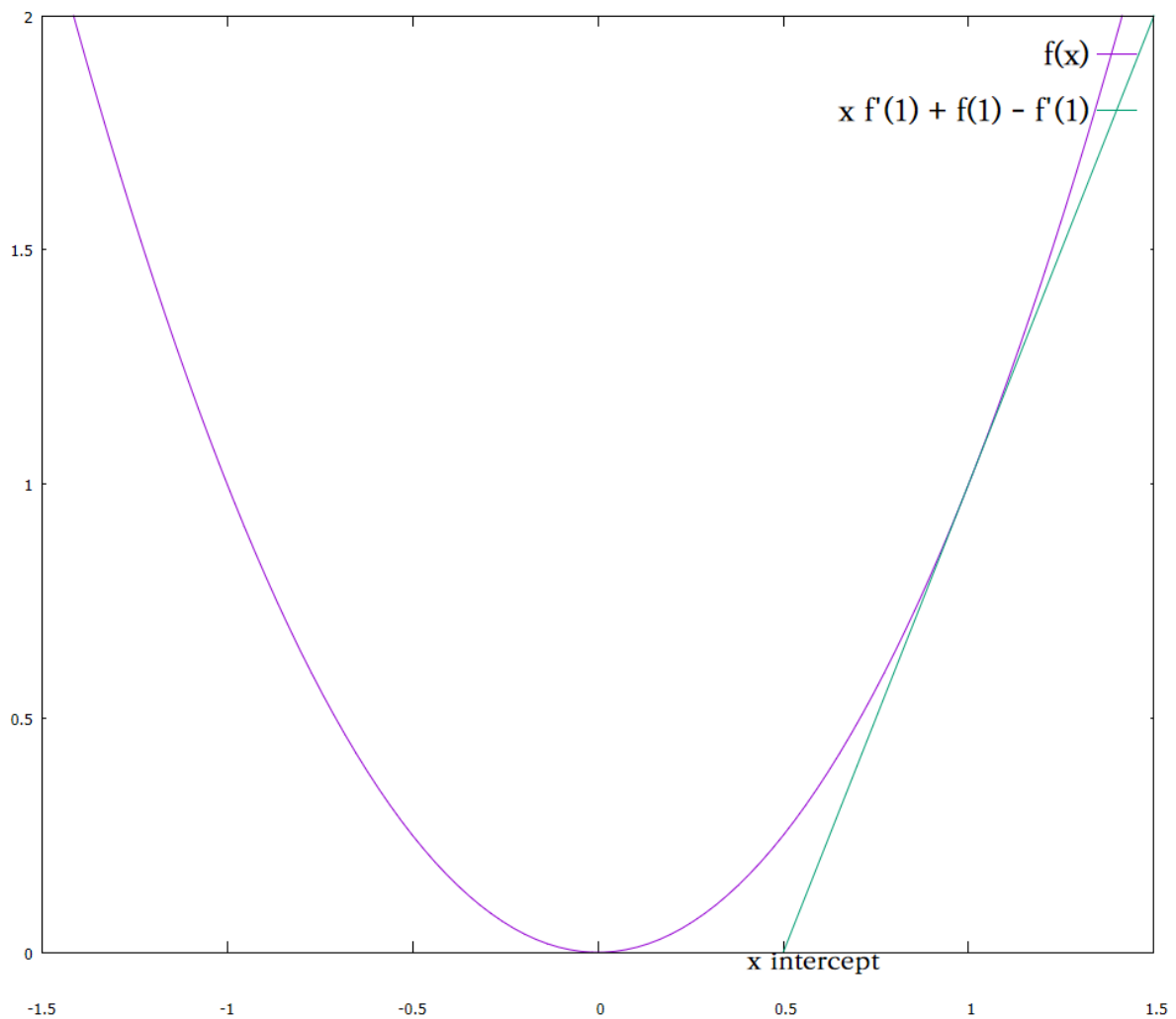


Figure 2: 함수  $f(x)$  의  $x = 1$  에서의 접선  $l$  의  $x$  절편

이러한 과정을 오차  $|f(x_i)|$ 가 허용 오차  $\varepsilon$  보다 작아질 때까지 반복하여 근사해를 구한다.

#### 4. 알고리즘 설명

이제 주어진 코드를 첫 줄부터 살펴보자.

1. `x2 = number * 0.5F;`

입력 number 을 실수 0.5 로 나눈 값을  $x_2$  에 저장한다.

2. `y = number`

변수  $y$  에 입력 number 을 저장한다.

3. `i = * ( long * ) &y;`

실수형 변수  $y$  를 정수형으로 인식시킨 값을 변수  $i$  에 저장한다.

우리가 구하고자 하는 것은 제곱근의 역수이므로  $0 < y$  라고 가정하자.  $y$  를 이루는 비트들을 위와 같은 방법으로  $s, a, b$  로 나눈다면  $i = a^{2^3} + b$  임을 알 수 있다.

4. `i = 0x5f3759df - ( i >> 1 );`

0x5f3759df 는 16 진수 5f3759df, 즉 10 진수 1597463007 을 의미하며,  $i \gg 1$  은  $i$  를 이루는 모든 비트를 오른쪽으로 한 칸 이동한 것을 의미한다. 정수형 변수에서  $i \gg n$  은  $\lfloor \frac{i}{2^n} \rfloor$  과 같으므로, 이 줄은 변수  $i$  에 1597463007 에서  $\frac{i}{2}$  를 뺀 값을 다시 저장한다.

그러면 위의 계산에 따라

$$y = [1 + b \times 2^{-23}] \times 2^{a-127}$$

$$\log_2(y) = \log_2(1 + 2^{-23}b) + a - 127$$

이다. 이 때 항상  $0 \leq 2^{-23}b < 1$  이다.

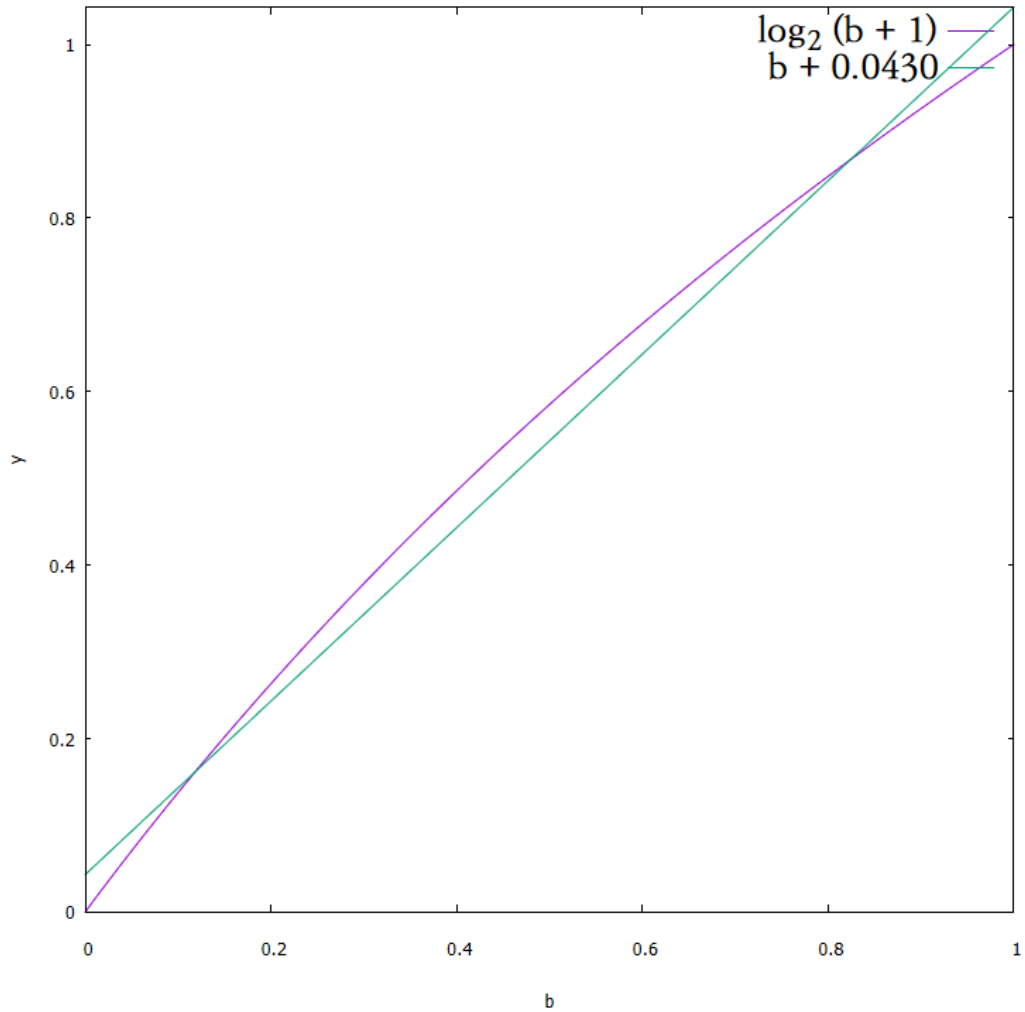


Figure 3:  $[0, 1]$  범위에서  $\log_2(b + 1)$  과  $b + 0.043$  의 그래프

위 그래프에서 보이듯,  $b \in [0, 1]$  일 때  $\log_2(b + 1)$  은  $b + 0.043$  와 매우 가까운 값을 가지므로  $b + 0.043$  으로 근사할 수 있다. 따라서

$$\log_2(y) \simeq 2^{-23}b + a - 127 + 0.043$$

$$= 2^{-23}i - 127 + 0.043$$

이다. 이제  $x = y^{-\frac{1}{2}}$  로 놓고,  $x$  를 이루는 비트들을 위와 같이  $s, c, d$  로 나눈 뒤  $j$  를  $x$  를 정수로 인식시킨 값이라 하자. 그러면

$$\begin{aligned}\log_2(x) &\simeq 2^{-23}j - 127 + 0.043 \\ &= -\frac{1}{2}[2^{-23}i - 127 + 0.043]\end{aligned}$$

이므로

$$\begin{aligned}2^{-23}j - 127 + 0.043 &= -\frac{1}{2}[2^{-23}i - 127 + 0.043] \\ \Rightarrow 254 - 0.086 - 2^{-22}j &= 2^{-23}i - 127 + 0.043 \\ \Rightarrow 381 - 0.129 - 2^{-23}i &= 2^{-22}j \\ \Rightarrow 381 \times 2^{22} - 0.129 \times 2^{22} - \frac{1}{2}i &= j \\ \Rightarrow 1\,597\,488\,759 - \frac{1}{2}i &\simeq j\end{aligned}$$

이다. 여기서 구해진 1597488759 는 1597463007 과 매우 가까운 값이며,  $x + C$  를  $\log_2(b+1)$  에 근사시키기 위해 사용한 상수  $C$  가 약간 달라 근소한 차이가 발생했을 것이라 예상할 수 있다.

5. `* ( float * ) &i;`

변수  $y$  에 정수형 변수  $i$  를 실수로 인식시킨 값을 저장한다.

6. `y = y * (threehalfs - (x2 * y * y));`

`threehalfs` 는 실수 상수 1.5 이다. 이 줄은 변수  $y$  에  $y\left(\frac{3}{2} - \frac{x_2 y^2}{2}\right)$  를 저장한다.

주어진  $y$  는 이제  $(\text{number})^{-\frac{1}{2}}$  에 매우 가까운 값이지만, 여전히 오차가 존재한다. 이 줄에서는 오차를 최소화하기 위해  $y$  에 뉴턴법을 1 회 적용한다. 뉴턴법을 적용하기 위한 과정은 다음과 같다.

상수  $C$  에 대해  $x = C^{-\frac{1}{2}}$  이면 방정식

$$\frac{1}{x^2} - C = 0$$

이 성립한다. 방정식의 좌변을  $f(x)$  라 하면

$$f'(x) = -\frac{1}{2x^3}$$

이고, 따라서  $x = x_0$  에서  $f(x)$  의 접선의 방정식은

$$\begin{aligned}0 &= -\frac{2x}{x_0^3} + \frac{2}{x_0^2} + \left(\frac{1}{x_0^2} - C\right) \\ &= -\frac{2x}{x_0^3} + \frac{3}{x_0^2} - C \\ &= \frac{3x_0 - 2x}{x_0^3} - C\end{aligned}$$

이며,  $x$  절편은

$$x = x_0 \left( \frac{3}{2} - x_0^2 \frac{C}{2} \right)$$

이다. 첫 줄에서  $x_2 = \frac{\text{number}}{2}$  로 정의했으므로

$$(\text{number})^{-\frac{1}{2}} \simeq y \left( \frac{3}{2} - \frac{x_2 y^2}{2} \right)$$

가 된다. 즉, 우리는  $(\text{number})^{-\frac{1}{2}}$  의 근사값을 계산했다.

7. `return y;`

마지막으로  $y$  의 값을 반환한다. 우리는 위에서  $y \simeq (\text{number})^{-\frac{1}{2}}$  임을 보였다.

## 5. 응용

고속 역제곱근 알고리즘이 가장 많이 응용되는 분야는 컴퓨터 그래픽이다. 예시로 어떤 벡터  $v = (v_1, v_2, v_3)$  이 주어졌을 때  $v$  와 방향이 같은 단위벡터  $u$  를 찾으려 한다고 가정하자. 그러면

$$u = \frac{v}{\|v\|}$$

이며, 이 때

$$\|v\| = \sqrt{v_1^2 + v_2^2 + v_3^2}$$

이므로,  $\text{number} = v_1^2 + v_2^2 + v_3^2$  로 놓고  $\text{number}$  에 고속 역제곱근 알고리즘을 취한 값을  $y$  라 하면  $u = yv$  로 간단히 나타낼 수 있다.

단위 벡터를 구하는 것은 컴퓨터 그래픽에서 매우 중요하고 기본적이며 자주 쓰이는 연산으로 이 연산의 실행 속도를 개선함으로써 그래픽 성능을 크게 향상시킬 수 있다. 테일러 급수를 이용한 기존의 방법에 비해 고속 역제곱근 알고리즘은 매우 빠르게 계산되므로, 이 방법으로 컴퓨터 게임 업계는 본격적인 3D 그래픽 시대를 열 수 있었다. 현대의 CPU 에는 역제곱근을 계산하는 기능이 내장되어 있으므로 더이상 프로그래머가 그것을 직접 구현할 필요가 없어져 사용되지 않지만, 이 알고리즘은 역사적으로 매우 중요한 의미를 갖는다고 할 수 있다.

## 부록

```
const mult32:f32 = 12102203.161561485;
const adder32: f32 = 1064872507.1615615;

fn fast_exp32(x: f32) -> f32 {
    union U {
        f: f32,
        i: i32,
    }

    unsafe {
        let mut u = U { f: x };
        u.i = (u.f * mult32 + adder32) as i32;
        u.f
    }
}
```

같은 방법으로 설계한, 지수함수를 빠르게 계산하는 알고리즘

위 코드에 대해 설명하면,

$x, y$  는  $e^x = y$  를 만족하는 Float32 타입의 수,  $i, j$  는 각각  $x, y$  의 각 비트를 보존한 채 Int32 로 인식시킨 수라고 하자.  $b \in [0, 1]$  일 때  $\log_2(1 + b)$  를  $b + 0.0573$  으로 근사시킬 수 있으므로

$$\begin{aligned}
 x &= \ln y \\
 &= \frac{\log_2 y}{\log_2 e} \\
 &\simeq \frac{1}{\log_2 e} [2^{-23}j - 127 + 0.0573]
 \end{aligned}$$

이다. 우변을 정리하여

$$2^{23}(x \log_2 e + 127 - 0.0573) = j$$

을 얻을 수 있다. 따라서,

$$y \simeq * (\text{Float32} *) (2^{23}(x \log_2 e + 127 - 0.0573) \text{ as Int32})$$

이 된다. 따라서 위와 같은 코드로 지수함수를 계산할 수 있다. 역제곱근과 마찬가지로 현대의 CPU 는 지수함수를 하나의 어셈블리 명령어로 계산하므로 의미 있는 알고리즘은 아니다.

## 참고문헌

Lomont, C. (2003). Fast inverse square root. Tech-315 nical Report, 32.  
 Burden, R. L. (2011). Numerical analysis. Brooks/Cole Cengage Learning.