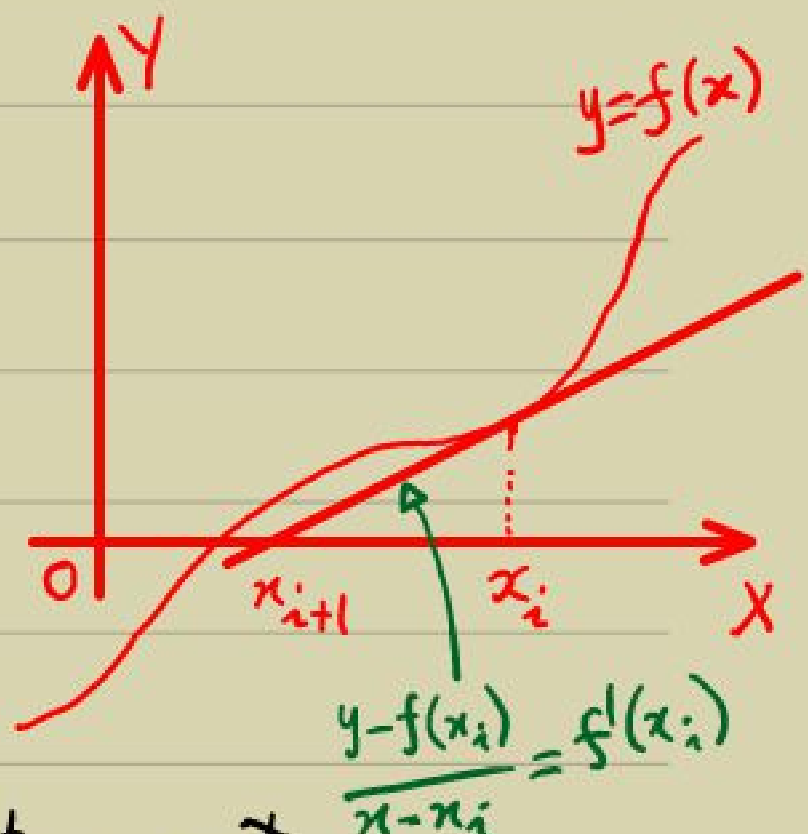
Fast integer division

- The question of computing a/b up to some decimal places reduces to that of computing $1/b$.

- If b is ℓ-bits & we want to compute $1/b$ upto ℓ places, then the school method (long division) requires $O(\ell^2)$ time.

- We could apply fast integer multiplication here to get an $\tilde{O}(\ell)$ time algorithm!

Newton's Approximation:
(1685)



- It is an iterative way to find roots of a function $y=f(x)$.
- Starts with a suitable point $(x_0,0)$ & gets closer to a root.

- Algorithm:

(1) Compute $x_{i+1} = x_i - \dfrac{f(x_i)}{f'(x_i)}$

for $i = 0, 1, \ldots$

till $|f(x_{i+1})|$ is small enough.

(2) Output $x_0, x_1, x_2, \ldots$ as approximations to a root of $f(x)$.

- In the case of integer division the relevant curve is $y = f(x) := \frac{1}{x} - b$, where $b \in \mathbb{N}$.

▷ $f'(x) = -1/x^2$.

▷ Newton's iteration becomes $x_{i+1} = x_i - \dfrac{x_i^{-1} - b}{-x_i^{-2}}$.

$\Rightarrow x_{i+1} = x_i(2 - b x_i)$.

- Let $2^{\ell-1} \leq b < 2^{\ell}$ & $x_0 := 2^{-\ell}$.

- <u>Lemma</u>: $\forall i,\ |x_i - b^{-1}| \leqslant 1/b \cdot 2^{2^i}$.

<u>Pf</u>: • $i=0$: $|x_0 - b^{-1}| = \left|2^{-\ell} - \frac{1}{b}\right| = \frac{|b-2^\ell|}{b \cdot 2^\ell} \leqslant \frac{2^{\ell-1}}{b \cdot 2^\ell} \leqslant \frac{1}{2b}$

• Let us assume it to hold up to i.

$|x_{i+1} - b^{-1}| = \left|2x_i - b x_i^2 - \frac{1}{b}\right| = b \cdot \left|x_i - \frac{1}{b}\right|^2 \leqslant b \cdot \frac{1}{b^2 \cdot 2^{2^{i+1}}}$

$= 1/b \cdot 2^{2^{i+1}}$. □

- Thus, to know the value of $1/b$ up to $O(\ell)$ places, it suffices to iterate up to $i = O(\lg \ell)$.

- <u>Complexity analysis</u>: If we use integer multiplication with time complexity $M(m)$, then computing b^{-1} (up to $O(\ell)$ places) requires time:

$$\sum_{1 \leqslant i \leqslant \lg \ell} M(2^i) \leqslant M\left(\sum_i 2^i\right) \leqslant M(2\ell) = \tilde{O}(\ell).$$

precision: $|1-bx_i|$'s bit-length doubles in an iteration

- Recall gcd computation: It's j-th step is $r_{j-2} - q_j r_{j-1} = r_j$ with $r_{-1} = a$, $r_0 = b$.

$\Rightarrow$ The time complexity of gcd is:

$$\sum_{j\in[i]} M(\lg q_j)$$

$\because M$ is superlinear $\rightarrow$

$$\leq M\left(\sum_{j=1}^{i} \lg q_j\right) \leq M\left(\sum_{j=1}^{i} (\lg r_{j-2} - \lg r_{j-1})\right)$$

$$\leq M(\lg a).$$

$\triangleright$ Similar complexity for computing $a^{-1} \bmod b$ & doing Chinese remaindering.

<u>Revisit integer multiplication</u>

- Recall that $\hat{a}(x)$, $\hat{b}(x)$ & $\hat{a}\cdot\hat{b}$ are polynomials of degree $< m$.

  Also, coeffs. of $\hat{a}, \hat{b}$ are $< 2^k$ & those of $\hat{a}\cdot\hat{b}$ are $< m\cdot 2^{2k}$.

- We could work over the ring

$$R := \mathbb{Z}/\langle m\cdot(2^{2k}+1)\rangle .$$

- $\omega := 4 \bmod \langle 2^{2k}+1\rangle$ has order $2k > m =: 2^u$.

- Since $(m, 2^{2k}+1)=1$ we can do all the computations $\bmod\langle m\rangle$, $\bmod\langle 2^{2k}+1\rangle$ & combine the two by Chinese remaindering.

▷ Computation of $\hat{a}\cdot\hat{b} \bmod m$ can be easily done in $O(\ell)$-time. $\because m=2^u$ is a 2-power.

- Computation of $\hat{a}\cdot\hat{b} \bmod \langle 2^{2k}+1\rangle$ is recursion heavy. We get the recurrence:

$$T(\ell) = m\cdot T(2k) + O(\ell\cdot \lg \ell)$$

$$\Rightarrow \quad T'(\ell) = 2\cdot T'(2\sqrt{\ell}) + O(\lg \ell),$$

where $T' := T(\ell)/\ell$.

▷ It gives $T'(\ell) = O(\lg\ell\cdot \lg\lg\ell)$.

- Chinese remaindering reduces to 2 mult. as we already have: $(2^u)^{-1} \equiv -2^{2k-u} \bmod \langle 2^{2k}+1\rangle$ &

$$(2^{2k}+1)^{-1} \equiv 1 \bmod \langle 2^u\rangle .$$