- What if R has $\underline{\text{char} = 2}$?

- We could take $\ell = 3^n$, devise a virtual ℓ-th root of unity w & apply DFT[w].

Theorem: In all cases, $h = f \cdot g$ in $R[x]$ can be computed in $O(\ell \cdot \lg \ell \cdot \lg\lg \ell)$ R-operations.

Fast Integer Multiplication

- Say, $a, b \in \mathbb{N}$ are $\ell = 2^n$ bit numbers.

- Let $k := 2^{\lceil n/2 \rceil}$, $m := 2^{\lfloor n/2 \rfloor}$. So, $km = \ell$.

- Idea: Reduce the $a \cdot b$ computation to the multiplication of polynomials $\hat{a}(x) \cdot \hat{b}(x)$, where $\hat{a}, \hat{b}$ are of degree $\leq m$.

- Write a as $\sum_{i=0}^{m-1} \hat{a}_i \cdot 2^{ki}$, $\hat{a}_i \in \mathbb{N}$

  and b as $\sum_{i=0}^{m-1} \hat{b}_i \cdot 2^{ki}$, $\hat{b}_i \in \mathbb{N}$.

▷ Clearly, $\hat{a}_i, \hat{b}_i < 2^k$.

- Define integral polynomials,
  $\hat{a}(x) := \sum_{0 \le i \le m-1} \hat{a}_i \cdot x^i$ &

  $\hat{b}(x) := \sum_{i} \hat{b}_i \cdot x^i$.

▷ $\hat{a}, \hat{b}$ have degree m & their coefficients are k-bits.

▷ The coefficient of x^j in $\hat{a}(x) \cdot \hat{b}(x)$ is $\sum_{0 \le i \le j} \hat{a}_i \cdot \hat{b}_{j-i}$, whose magnitude $< m \cdot 2^{2k} < 2^{3k}$.

- Idea: We compute the polynomial product $\hat{a}(x) \cdot \hat{b}(x) \pmod{2^{3k}+1}$. Finally, compute at $x = 2^k$.

▷ The ring $R := \mathbb{Z}/\langle 2^{3k}+1 \rangle$ has a $(2k)$-th root of unity $w := 8$. (Note: $2k \geq 2m$)

- So, we can follow the polynomial multiplication algorithm based on DFT[8]:

(1) Compute DFT[w] of $\hat{a}(x)$ & $\hat{b}(x)$ in $R[x]$.
(2) Compute the $2m$ products $\hat{a}(w^i) \cdot \hat{b}(w^i)$ in R.
(3) Compute DFT[w^{-1}] of $\{\hat{c}(w^i) \mid i\}$, yielding $\hat{c}(x)$.
(4) Output $\hat{c}(2^k)$.

Complexity analysis:

Steps (1) & (3): By the fast DFT algorithm it can be done in $O(m \lg m)$ R-operations, which means $O(km \lg m)$ time. (R-mult. needed are the ones by w^i.)

Step(2): We need to do $2m$ multiplications of $(3k)$-bit numbers. So, we get the recurrence:

$$T(\ell) = 2m \cdot T(3k) + O(\ell \cdot \lg m).$$

$\Rightarrow$

$$T(\ell) = O(\ell \cdot \lg^{\alpha} \ell), \text{ where } \alpha := \log_2 6.$$

Step(4): This is simply rewriting the number in bits. So, time is $O(\ell)$.

Theorem: Two ℓ-bit integers can be multiplied in $O(\ell \cdot \lg^{\alpha} \ell)$ time, $\alpha := \log_2 6$.

- The fastest integer multiplication algorithm known is due to Fürer (2007). Its time complexity is $\ell \cdot \lg \ell \cdot 2^{O(\lg^* \ell)}$, where $\lg^* \ell$ is defined to be the least number i s.t. $\underbrace{\lg \lg \ldots \lg}_{i \text{ times}} \ell < 2$

- The idea is to use multivariates & a much smaller R.

Fast integer division

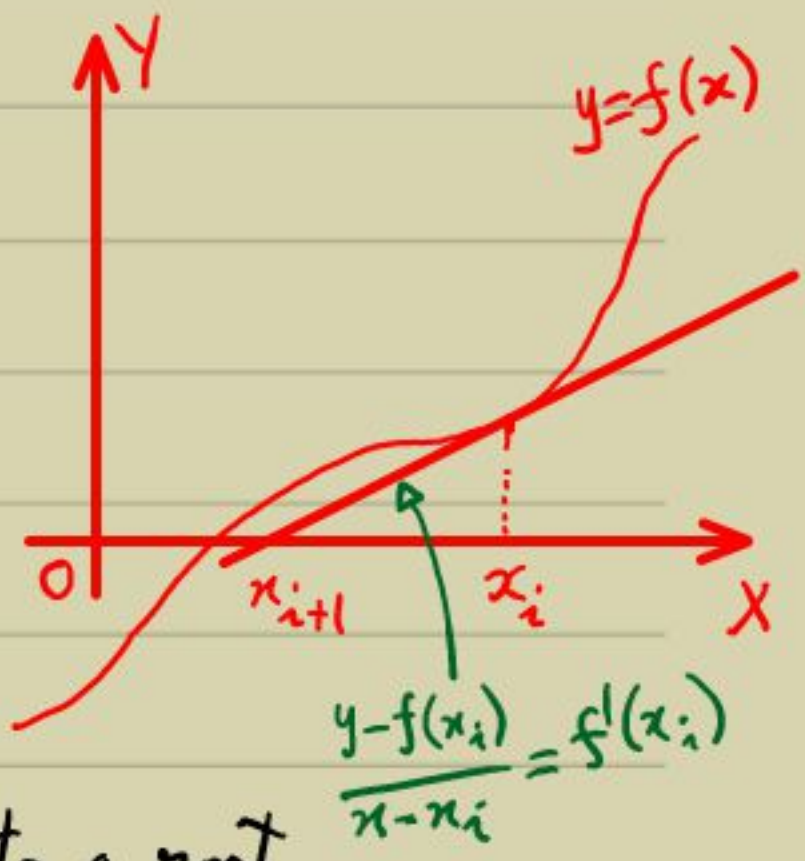
- The question of computing a/b up to some decimal places reduces to that of computing $1/b$.

- 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:

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

- Lemma: $\forall i,\ |x_i - b^{-1}| \leq 1/b \cdot 2^{2^i}$.

Pf: • $i=0$: $|x_0 - b^{-1}| = \left|2^{-\ell} - \frac{1}{b}\right| = \frac{|b-2^\ell|}{b\cdot 2^\ell} \leq \frac{2^{\ell-1}}{b\cdot 2^\ell} \leq \frac{1}{2b}$

• Let us assume it to hold up to i.

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

$$= 1/b\cdot 2^{2^{i+1}}.$$ □

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

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

$$\sum_{1\leq i\leq O(\lg \ell)} M(\ell + 2^i) = O(M(\ell)\cdot \lg \ell)$$

$$= \tilde{O}(\ell).$$

↑ ($b^{-1} - x_i$)'s bit-length doubles in an iteration