

Lecture Notes on Semi-numerical Algorithms

Basics

Let Z be the set of integers and N be the set of natural numbers. That is,

$$\begin{aligned}Z &= \{ \dots, -2, -1, 0, 1, 2, \dots \} \\N &= \{ 0, 1, 2, \dots \}\end{aligned}$$

In the following, variables take integers by default.

Theorem. For any integer a and any integer $b > 0$, there are unique q and r such that

$$a = qb + r, 0 \leq r < b$$

Here q is the quotient and r is the remainder. If $r = 0$, that is, if b divides a , we write $b \mid a$.

If $m \mid a-b$ for $m > 0$, we write $a \equiv b \pmod{m}$, and say a is congruent to b modulo m .

Lemma. For fixed m , the relation “ $\equiv \pmod{m}$ ” is an equivalence relation, that is,

- (1) reflexive, $a \equiv a \pmod{m}$
- (2) symmetric, $a \equiv b \pmod{m} \Rightarrow b \equiv a \pmod{m}$
- (3) transitive, $a \equiv b \pmod{m} \wedge b \equiv c \pmod{m} \Rightarrow a \equiv c \pmod{m}$.

By this lemma, we can partition Z by the relation “ $\equiv \pmod{m}$ ”, or simply “ $\equiv (m)$ ”, into equivalence classes. Let $[a]$ be the equivalence class including a , that is

$$[a] = \{ b \mid b = a + km, k \in Z \}.$$

Let the operation on equivalence classes be defined by

$$[a] \pm [b] = [a \pm b]$$

$$[a] * [b] = [a*b].$$

These operations are well defined thanks to the following lemma. Then the quotient algebra $Z/\equiv (m)$ forms a finite ring of order m . We express $Z/\equiv (m)$ by representatives $\{0, 1, \dots, m-1\}$. We also denote $Z/\equiv (m)$ by Z_m .

Example. Z_5 .

+	0, 1, 2, 3, 4
0	0 1 2 3 4
1	1 2 3 4 0
2	2 3 4 0 1
3	3 4 0 1 2
4	4 0 1 2 3

*	0 1 2 3 4
0	0 0 0 0 0
1	0 1 2 3 4
2	0 2 4 1 3
3	0 3 1 4 2
4	0 4 3 2 1

Lemma. Let $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$. Then

$$a \pm c \equiv b \pm d \pmod{m}, a * c \equiv b * d \pmod{m}.$$

Let $\gcd(a, b)$ be the greatest common divisor of $a > 0$ and $b > 0$. If $\gcd(a, b) = 1$, a and b are said to be mutually prime.

Lemma. For any $a > 0$ and $b > 0$, there exist x and y such that $d = ax + by$ where $d = \gcd(a, b)$.

From this we have,

Lemma. For $a > 0$ and $m > 0$, $\gcd(a, m) = 1$ if and only if for some x , $a * x \equiv 1 \pmod{m}$.

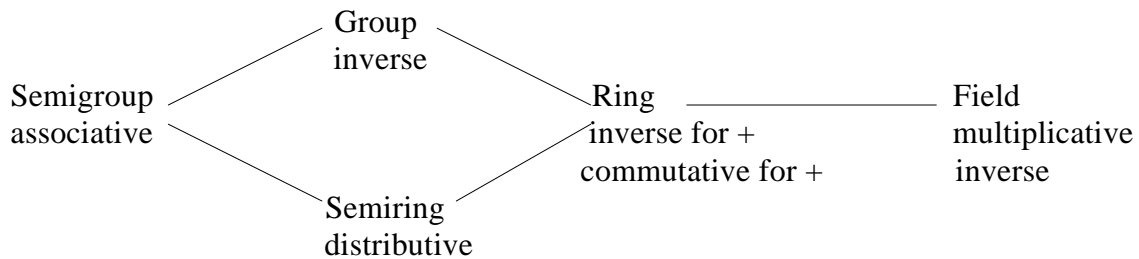
Corollary. Let $p > 0$ be a prime number. Then $Z/\equiv (p)$ or Z_p is a field.

Note. In a field F , any $a \in F$ has a multiplicative inverse a^{-1} , that is, $a * a^{-1} = 1$.

For algebraic systems, we have the following axioms.

Associative	$a(bc) = (ab)c$
Commutative	$ab = ba$
Unit element	$1a = a1$
Inverse	$a a^{-1} = a^{-1} a = 1$
Distributive	$a(b+c) = ab + ac, (a+b)c = ac + bc.$

We have weaker systems to stronger systems from left to right with additional axioms.



Horner's algorithm for evaluating polynomials

If we write a polynomial of degree n in any programming language for some specific n like

$$y = 5*x*x*x*x + 3*x*x*x + 4*x*x + 1*x + 2 \quad (n=4),$$

it would take $O(n^2)$ time, as we need $n + (n-1) + \dots + 1 + 0$ multiplications. Let us convert the above form into

$$y = (((5*x + 3)*x + 4)*x + 1)*x + 2.$$

Then obviously we can finish the evaluation in $O(n)$ time. Let $a[i]$ ($i=0, \dots, n-1$) be coefficients for the term of the i -th degree. We can formalize Horner's algorithm as follows:

```
y:=a[n];
for i:=n-1 downto 0 do y:=y*x + a[i];
```

Evaluation of x^n

In stead of performing $n-1$ multiplications, we can compute x^n in $2\lfloor \log n \rfloor$ multiplications.

```
n = (b(m-1)...b(1)b(0)) : binary expression of n, e.g., 5 = (101)
y:=1;
for i:=m-1 downto 0 do begin
  y:=y*y;
  if b(i)=1 then y:=y*x
end.
```

This method is called "repeated squaring".

Theory of Fibonacci Numbers

The Fibonacci sequence is defined by

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \dots$$

That is

$$x(0) = 0, \quad x(1) = 1, \\ x(n+2) = x(n+1) + x(n), \quad (n \geq 0).$$

A homogeneous difference equation is defined by

$$x(0)=b(0), x(1)=b(1), \dots, x(k-1)=b(k-1),$$

$$x(n+k) = a(1)x(n+k-1) + \dots + a(k)x(n).$$

The general solution is given by

$$x(n) = c(1)r(1)^n + \dots + c(k)r(k)^n,$$

where $r(i)$ is the i -th root of the characteristic equation

$$r^k - a(1)r^{k-1} - \dots - a(k) = 0.$$

Here we assume that all the k roots of the characteristic equation are different. The constants $c(1), \dots, c(k)$ are determined by the k initial conditions.

For the Fibonacci sequence, the characteristic equation is

$$r^2 - r - 1 = 0,$$

the solution of which is given by

$$r(1) = (1 + \sqrt{5})/2, r(2) = (1 - \sqrt{5})/2.$$

Therefore the general solution becomes as follows.

$$x(n) = c(1)r(1)^n + c(2)r(2)^n.$$

From the initial condition, we have the following simultaneous equation.

$$c(1) + c(2) = 0$$

$$c(1)(1 + \sqrt{5})/2 + c(2)(1 - \sqrt{5})/2 = 1.$$

From this we have

$$c(1) = 1/\sqrt{5}, c(2) = -1/\sqrt{5}.$$

Thus the n -th Fibonacci number $x(n)$ is given by

$$x(n) = (1/\sqrt{5})((1 + \sqrt{5})/2)^n - (1/\sqrt{5})((1 - \sqrt{5})/2)^n$$

The growth rate of $x(n)$ is measured by the first term which is dominant, that is, an exponential function of $r(1) = 1.618..$

With a primitive approach it takes $O(n)$ time to compute $x(n)$. The following algorithm computes $x(n)$ in $O(\log n)$ time, which is based on repeated squaring. The difference equation is expressed by vectors and a matrix as

$$(x(0) \ x(1)) = (0, 1)$$

$$(x(n-1) \ x(n)) = (x(n-2) \ x(n-1)) \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Then we have

$$(x(n-1) \ x(n)) = (x(0) \ x(1)) \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{(n-1)}$$

Hence we can use the fast algorithm to evaluate x^n based on repeated squaring as applied to a matrix.

Euclidean Algorithm for Greatest Common Divisors

We compute the greatest common divisor of a and $b > 0$, $\gcd(a, b)$. Let us divide a by b , yielding the following equation for the quotient q and remainder r .

$$a = bq + r, \quad 0 \leq r < b$$

We observe that $\gcd(a, b) = \gcd(b, r)$. The important point here is that the problem of $\gcd(a, b)$ has been reduced to that of $\gcd(b, r)$. Then we can repeat this process as shown below.

Let $a = r(0)$, $b = r(1)$, $q = q(1)$ and $r = r(2)$ in the above. We repeat division as follows:

$$\begin{aligned} a &= b \cdot q(1) + r(2), & 0 \leq r(2) < b \\ b &= r(2) \cdot q(2) + r(3), & 0 \leq r(3) < r(2) \\ &\dots \\ r(i-1) &= r(i) \cdot q(i) + r(i+1), & 0 \leq r(i+1) < r(i) \\ &\dots \\ r(n) &= r(n) \cdot q(n) + r(n+1), & r(n+1) = 0 \end{aligned}$$

That is we finish at the n -th division.

Theorem. $\gcd(a, b) = r(n)$.

Proof. Theorem follows from

$$\gcd(a, b) = \gcd(b, r(2)) = \gcd(r(2), r(3)) = \dots = \gcd(r(n), r(n+1)) = \gcd(r(n), 0) = r(n).$$

Example. $a = 91$, $b = 35$.

$$\begin{aligned} 91 &= (35)2 + 21, & q(1) &= 2, r(2) = 21 \\ 35 &= (21)1 + 14, & q(2) &= 1, r(3) = 14 \\ 21 &= (14)1 + 7, & q(3) &= 1, r(4) = 7 \\ 14 &= (7)2 + 0, & q(4) &= 2, r(5) = 0 \end{aligned}$$

$$\gcd(91, 35) = 7.$$

Sequences $\{a(i)\}$ and $\{b(i)\}$ are defined as follows:

$$\begin{aligned} a(0) &= 0, & a(1) &= 1, & a(i) &= a(i-2) - q(i-1)a(i-1) \\ b(0) &= 1, & b(1) &= 0, & b(i) &= b(i-2) - q(i-1)b(i-1). \end{aligned}$$

Theorem. For any $i > 0$, $a*b(i) + b*a(i) = r(i)$.

Proof. Induction on i .

$$\begin{aligned} i=0, & \quad a*b(0) + b*a(0) = a = r(0) \\ i=1, & \quad a*b(1) + b*a(1) = b = r(1) \end{aligned}$$

Assume theorem is true for up to i . Then

$$\begin{aligned} a*b(i+1) + b*a(i+1) &= a(b(i-1) - q(i)b(i)) + b(a(i-1) - q(i)a(i)) \\ &= a*b(i-1) + b*a(i-1) - q(i)(a*b(i) + b*a(i)) \\ &= r(i-1) - q(i)r(i) \\ &= r(i-1). \end{aligned}$$

Let $i = n$ in the theorem. Then we have

$$a*b(n) + b*a(n) = \gcd(a, b).$$

That is we can find an integer solution (x, y) that satisfies the equation

$$ax + by = \gcd(a, b).$$

Example. For the previous example of $a = 91$ and $b = 75$. We have

$$\begin{aligned} a(2) &= -2, & a(3) &= 3, & a(4) &= -5 \\ b(2) &= 1, & b(3) &= -1, & b(4) &= 2. \end{aligned}$$

From this we have the solution $(x=1, y=-5)$ for $91x + 35y = 7$.

In the case of $\gcd(a, b) = 1$, the solution for $ax + by = 1$ has the following meaning.

$$x = a^{(-1)} \bmod b, \quad y = b^{(-1)} \bmod a.$$

Example. If we divide both sides of the previous example, we have $13x + 5y = 1$. Since 13 and 5 are mutually prime, each has a multiplicative inverse modulo each other. For example, $5^{(-1)} = -5$. To get a positive multiplicative inverse, we have $-5 + 13 = 8$.

The following procedure computes the greatest common divisor of a and b together with the sequences $\{a(i)\}$ and $\{b(i)\}$.

1. $w_0 := a; w_1 := b;$
 2. $u_0 := 0; u_1 := 1;$
 3. $v_0 := 1; v_1 := 0;$
 4. while $w_1 > 0$ do begin
 5. $q := w_0 \text{ div } w_1;$
 6. $w_2 := w_0 - q * w_1; w_0 := w_1; w_1 := w_2;$
 7. $u_2 := u_0 - q * u_1; u_0 := u_1; u_1 := u_2;$
 8. $v_2 := v_0 - q * v_1; v_0 := v_1; v_1 := v_2$
 9. end
- { $w_0 = \text{gcd}(a, b), u_0 = b^{-1} \text{ mod } a, v_0 = a^{-1} \text{ mod } b$ }.

If u_0 or v_0 is negative at the end, we can add a or b respectively to get a positive inverse.

Analysis of Euclidean algorithm

Let $D(a, b)$ be the number of divisions performed in the algorithm. Let $f(n)$ be defined by

$$f(0) = 1, f(1) = 2,$$

$$f(n) = f(n-1) + f(n-2), \quad n \geq 2$$

That is, $f(n) = x(n+2)$ where $x(n)$ is the n -th Fibonacci number. Thus

$$f(n) = (1/\sqrt{5})((1+\sqrt{5})/2)^{n+2} - (1/\sqrt{5})((1-\sqrt{5})/2)^{n+2}.$$

Lemma.

$$\text{For all } n \geq 0, \quad f(n+1) \leq 2 * f(n), \quad f(n) \geq ((1+\sqrt{5})/2)^n$$

Theorem.

$$\text{For } 0 \leq b \leq a \leq f(n), \quad D(a, b) \leq n \text{ for } n \geq 0.$$

Proof. Induction on n .

Basis. $n=1$. Since $f(1) = 2$, we classify a and b such that $0 \leq b \leq a \leq 2$ into a few cases.

When $b=0$, $D(a, b)=0$. When $b=1$, $D(a, b)=1$. When $a=b=2$, $D(a, b)=1$.

Induction step. Assume theorem is true for up to n , and assume $0 \leq b \leq a \leq f(n+1)$. The we classify the situation into two cases.

(1) $b \leq f(n)$. When $b=0$, $D(a, b)=0$. Let $b > 0$. Since $0 \leq r(2) < b$, we have $D(a, r(2)) \leq n$ from the inductive hypothesis. Thus $D(a, b) = D(b, r(2)) + 1 \leq n+1$.

(2) $f(n) < b$. From lemma, we have $a < 2b$. Thus

$$r(2) = a \text{ mod } b = a - b < f(n+1) - f(n) = f(n+1).$$

When $r(2) = 0$, $D(a, b) = 1$. When $r(2) > 0$, we have $0 \leq r(3) < r(2) < f(n-1)$. From the inductive hypothesis,

$$D(a, b) = D(b, r(2)) + 1 = D(R(2), r(3)) + 2 < n-1 + 2 = n+1.$$

Now let a and b be positive integers expressed by up to k bits. Then we have $0 <= b <= a <= 2^k - 1$. Let $\phi = (1 + \sqrt{5})/2$ and $n = \lceil k \log_{\phi} 2 \rceil = 1.45k$. Then from $\phi^k >= 2^k$ and lemma, where ϕ is the first root of the characteristic equation, we have

$$0 <= b <= a < \phi^n <= f(n).$$

From theorem, we have $D(a, b) <= n$. That is, the number of divisions is not more than $1.44k$. Let the time for division of two k -bit numbers be $O(D(k))$. Then the Euclidean algorithm runs in $O(kD(k))$ time. If we use the simple $O(k^2)$ algorithm for division, the Euclidean algorithm runs in $O(k^3)$ time.

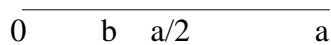
An easier analysis of Euclid's algorithm

In the above we showed that $D(a, b)$ is bounded by $1.45k$ if a and b are k -bit integers. If we are satisfied with the bound of $2k$, the following proof is easier.

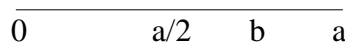
Lemma. Let $0 <= b < a$. Then $a \bmod b < a/2$.

Proof. Suppose $b <= a/2$. Then $a \bmod b < b <= a/2$. Suppose $b > a/2$. Then $a \bmod b = a - b < a/2$. See below.

First case



Second case



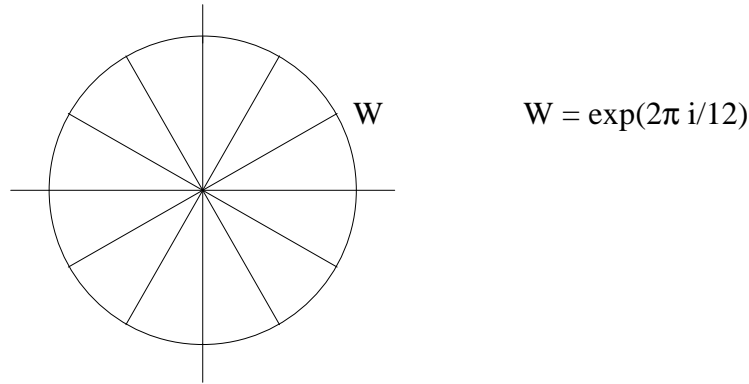
In the division process of Euclid's algorithm, we can halve the magnitude from $r(i)$ to $r(i+2)$. We can halve a at most $\log(a)$ ($<= k$) times. At each halving i is advanced by 2. Thus the number of divisions is bounded by $2k$. See below.

$$\begin{aligned} & \dots \\ r(i) &= r(i+1)q(i+1) + r(i+2) \\ r(i+1) &= r(i+2)q(i+2) + r(i+3) \\ r(i+2) &= r(i+3)q(i+3) + r(i+4) \\ & \dots \end{aligned}$$

The important observation here is we can compute a multiplicative inverse mod m of a k -bit number a in polynomial time of k , without factoring the numbers a and m . When k is a few hundred, this computation is within seconds, whereas factoring of a k -bit number for such k requires more than centuries on any high speed computer. This big difference of computing time is the basis of the security of the modern crypto-systems.

Integer and Polynomial Multiplications

Let W be the principal n -th root of 1 in the complex plane, that is, $W = \exp(2\pi i/n)$, where $i = \sqrt{-1}$. See below.



Then we have

$$(1) \quad W^n = 1$$

$$(2) \quad \sum_{j=0, n-1} W^{kj} = 0 \quad (k = 1, \dots, n-1)$$

The discrete Fourier transform (DFT) of the sequence of complex numbers $X(j)$ ($j=0, \dots, n-1$) is defined by the sequence $y(k)$ ($k=0, \dots, n-1$) where

$$y(k) = \sum_{j=0, n-1} x(j)W^{kj}. \quad k = 0, \dots, n-1$$

The inverse discrete Fourier transform (IDFT) of sequence $\{y(k)\}$ is defined by the sequence $z(l)$ where

$$z(l) = (1/n) \sum_{k=0, n-1} y(k)W^{-lk} \quad (l = 0, \dots, n-1)$$

Theorem. $x(j) = z(j)$ for $j = 0, \dots, n-1$. That is,

$$x(j) = (1/n) \sum_{k=0, n-1} y(k)W^{-kj}, \quad j=0, \dots, n-1.$$

Proof. Let $\mathbf{x} = (x(0), \dots, x(n-1))$, $\mathbf{y} = (y(0), \dots, y(n-1))$, and $\mathbf{z} = (z(0), \dots, z(n-1))$. Let matrices A and B be defined as $A = (a(i,j))$ and $B = (b(i,j))$ where

$$a(i,j) = W^{ij}, \quad b(i,j) = (1/n)W^{-ij}$$

Then we have $\mathbf{y} = \mathbf{x}A$ and $\mathbf{z} = \mathbf{y}B$. It suffices to show that $B = A^{-1}$. Let $C = BA$. Then we have

$$c(i,j) = (1/n) \sum_{k=0, n-1} W^{(j-i)k}.$$

When $i = j$, $W^{(j-i)k} = 1$, so $c(i,i) = 1$. When $i \neq j$, let $l = j - i$. Then from (2), we have

$$\sum_{k=0, n-1} W^{lk} = 0.$$

Thus $c(i,j) = 0$ for $i \neq j$.

From this we see that IDFT is indeed an inverse transform. Using FFT (Fast Fourier Transform) we can compute DFT and IDFT in $O(n \log n)$ time.

The convolution of two sequences $(x(0), \dots, x(n-1))$ and $(y(0), \dots, y(n-1))$ is defined by $(z(0), \dots, z(2n-1))$ where

$$z(j) = \sum_{k=0, n-1} x(j-k)y(k) \quad (j = 0, \dots, 2n-1).$$

Here we assume $x(k) = 0$ for $k < 0$ and $k \geq n$. From this definition, it is clear that $z(2n-1) = 0$, which is added to the sequence to adjust to the length of $2n$.

Theorem. Let the DFTs of $(x(0), \dots, x(n-1), 0, \dots, 0)$ and $(y(0), \dots, y(n-1), 0, \dots, 0)$ of length $2n$ each be $(x'(0), \dots, x'(2n-1))$ and $(y'(0), \dots, y'(2n-1))$. The convolution $(z(0), \dots, z(2n))$ defined above is equal to the IDFT of $(x'(0)y'(0), \dots, x'(2n-1)y'(2n-1))$.

Proof. Let the DFT of $(z(0), \dots, z(2n-1))$ be $(z'(0), \dots, z'(2n-1))$. Then

$$\begin{aligned} z'(l) &= \sum_{j=0, n-1} \sum_{k=0, n-1} x(j-k)y(k)W^{lj} \\ &= \sum_{k=0, n-1} \sum_{i=-k, 2n-1-k} x(i)y(k)W^{l(i+k)}, \quad \text{where } i = j-k \\ &= \sum_{k=0, n-1} \sum_{i=0, n-1} x(i)y(k)W^{l(i+k)} \\ &= \sum_{i=0, n-1} x(i)W^{li} \sum_{k=0, n-1} y(k)W^{lk}. \end{aligned}$$

On the other hand, for $(x'(0), \dots, x'(2n-1))$ and $(y'(0), \dots, y'(2n-1))$ we have

$$x'(l) = \sum_{k=0, 2n-1} x(k)W^{lk} = \sum_{k=0, n-1} x(k)W^{lk} \quad (l = 0, \dots, 2n-1)$$

$$y'(l) = \sum_{k=0, 2n-1} y(k)W^{lk} = \sum_{k=0, n-1} y(k)W^{lk} \quad (l = 0, \dots, 2n-1)$$

Thus we have $z'(l) = x'(l)y'(l) \quad (l = 0, \dots, 2n-1)$.

Using FFT, we can compute convolution in $O(n \log n)$ time in the following way. Compute DFT x' and y' in $O(n \log n)$ time. Compute z' from x' and y' in $O(n)$ time. Finally compute IDFT z from z' in $O(n \log n)$ time. Direct computation takes $O(n^2)$ time.

Next let us consider multiplying two polynomials

$$f(z) = x(0) + x(1)z + \dots + x(n-1)z^{(n-1)}$$

$$g(z) = y(0) + y(1)z + \dots + y(n-1)z^{(n-1)}.$$

The product is given by

$$f(z)g(z) = x(0)y(0) + (x(0)y(1) + x(1)y(0))z$$

$$+ \dots$$

$$+ (x(0)y(k) + x(1)y(k-1) + \dots + x(k)y(0))z^k$$

$$+ \dots$$

$$+ x(n-1)y(n-1)z^{(2n-2)} + 0z^{(2n-1)}.$$

That is, the k -th coefficient is the convolution $z(k)$. Thus polynomial multiplication can be done in $O(n \log n)$ time. Next consider multiplying two multi-precision binary numbers $(a(n-1)\dots a(1)a(0))$ and $(b(n-1)\dots b(1)b(0))$. As we see from the following figure, we can modify the convolution computation for multiplication. Let the binary form of the product be $(c(2n-1)\dots c(1)c(0))$.

	$a(n-1)$	\dots	$a(1)$	$a(0)$
	$b(n-1)$	\dots	$b(1)$	$b(0)$
	$a(n-1)b(0)$	$a(1)b(0)$	$a(0)b(0)$	
	$a(n-1)b(1)$	$a(0)b(1)$		
	$a(n-1)b(n-1)$	$a(0)b(n-1)$		
	$c(2n-1)$	$c(n-1)$	$c(1)$	$c(0)$

Here $c(i)$ ($i=0, \dots, 2n-1$) can be computed in the following way. Let $d(i)$ ($i=0, \dots, 2n-1$) be the convolution of $(a(0), \dots, a(n-1))$ and $(b(0), \dots, b(n-1))$. Then

```

for i:= 0 to 2n-1 do begin
  c(i) := mod 2;
  q(i) := c(i) div 2;
  c(i+1) := c(i+1) + q[i]
end.

```

If we go through FFT, all computations described in this section can be done in $O(n \log n)$ time, whereas straightforward methods take $O(n^2)$ time.