Análise de Algoritmos

Estes slides são adaptações de slides

do Prof. Paulo Feofiloff e do Prof. José Coelho de Pina.

Aula 3

Transformada rápida de Fourier

Secs 30.1 e 30.2 do CLRS e 5.6 do KT.

Produto de polinômios

Problema: Dados dois polinômios

$a(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$ e

$b(x) = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}$,

calcular o polinômio $p(x) = a(x) \cdot b(x)$.

Produto de polinômios

Problema: Dados dois polinômios

$a(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$ e

$b(x) = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}$,

calcular o polinômio $p(x) = a(x) \cdot b(x)$.

Lembre-se que

$p(x) = a(x) \cdot b(x) = c_0 + c_1 x + \cdots + c_{2n-2} x^{2n-2}$, onde

$$c_k \;=\; a_o b_k + a_1 b_{k-1} + \cdots + a_k b_0,$$

para $k = 0, 1, \ldots, 2n-2$.

Produto de polinômios

Problema: Dados dois polinômios

$a(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$ e

$b(x) = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}$,

calcular o polinômio $p(x) = a(x) \cdot b(x)$.

Lembre-se que

$p(x) = a(x) \cdot b(x) = c_0 + c_1 x + \cdots + c_{2n-2} x^{2n-2}$, onde

$$c_k = a_o b_k + a_1 b_{k-1} + \cdots + a_k b_0,$$

para $k = 0, 1, \ldots, 2n-2$.

O vetor c é a convolução de a e b,
às vezes denotada por $a \otimes b$.

Produto de polinômios

De novo há um algoritmo $\mathrm{O}(n^2)$ óbvio para calcular $p(x)$, ou seja, para calcular $c_0, c_1, \ldots, c_{2n-2}$.

Produto de polinômios

De novo há um algoritmo $\mathrm{O}(n^2)$ óbvio para calcular $p(x)$, ou seja, para calcular $c_0, c_1, \ldots, c_{2n-2}$.

Queremos algo melhor!

Produto de polinômios

De novo há um algoritmo $\mathrm{O}(n^2)$ óbvio para calcular $p(x)$, ou seja, para calcular $c_0, c_1, \ldots, c_{2n-2}$.

Queremos algo melhor! Queremos um algoritmo $\mathrm{O}(n \lg n)$!

Produto de polinômios

De novo há um algoritmo $\mathrm{O}(n^2)$ óbvio para calcular $p(x)$, ou seja, para calcular $c_0, c_1, \ldots, c_{2n-2}$.

Queremos algo melhor! Queremos um algoritmo $\mathrm{O}(n \lg n)$!

Qual é a ideia do algoritmo?

Produto de polinômios

De novo há um algoritmo $\mathrm{O}(n^2)$ óbvio para calcular $p(x)$, ou seja, para calcular $c_0, c_1, \ldots, c_{2n-2}$.

Queremos algo melhor! Queremos um algoritmo $\mathrm{O}(n \lg n)$!

Qual é a ideia do algoritmo?

Dados n pares $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$,
existe um único polinômio $q(x)$ de grau menor que n
tal que $q(x_i) = y_i$ para $i = 0, \ldots, n-1$.

Produto de polinômios

De novo há um algoritmo $\mathrm{O}(n^2)$ óbvio para calcular $p(x)$, ou seja, para calcular $c_0, c_1, \ldots, c_{2n-2}$.

Queremos algo melhor! Queremos um algoritmo $\mathrm{O}(n \lg n)$!

Qual é a ideia do algoritmo?

Dados n pares $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$,
existe um único polinômio $q(x)$ de grau menor que n
tal que $q(x_i) = y_i$ para $i = 0, \ldots, n-1$.

Prova na aula...

Produto de polinômios

De novo há um algoritmo $\mathrm{O}(n^2)$ óbvio para calcular $p(x)$, ou seja, para calcular $c_0, c_1, \ldots, c_{2n-2}$.

Queremos algo melhor! Queremos um algoritmo $\mathrm{O}(n \lg n)$!

Qual é a ideia do algoritmo?

Dados n pares $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$,
existe um único polinômio $q(x)$ de grau menor que n
tal que $q(x_i) = y_i$ para $i = 0, \ldots, n-1$.

Prova na aula...

Representações alternativas de polinômios de grau $n-1$:

- seus n coeficientes, ou
- seu valor em n pontos distintos.

Ideia do algoritmo $\mathrm{O}(n \lg n)$

Entrada: $a = (a_0, \ldots, a_{n-1})$ e $b = (b_0, \ldots, b_{n-1})$.

- Obter pares

$$(x_0, y_0^a), \ldots, (x_{2n-2}, y_{2n-2}^a) \text{ e } (x_0, y_0^b), \ldots, (x_{2n-2}, y_{2n-2}^b)$$

onde $x_i \neq x_j$ para $i \neq j$, e

$$y_i^a = a(x_i) \text{ e } y_i^b = b(x_i) \text{ para } i = 0, \ldots, 2n-2.$$

Ideia do algoritmo $\mathrm{O}(n \lg n)$

Entrada: $a = (a_0, \ldots, a_{n-1})$ e $b = (b_0, \ldots, b_{n-1})$.

- Obter pares

  $$(x_0, y_0^a), \ldots, (x_{2n-2}, y_{2n-2}^a) \text{ e } (x_0, y_0^b), \ldots, (x_{2n-2}, y_{2n-2}^b)$$

  onde $x_i \neq x_j$ para $i \neq j$, e

  $$y_i^a = a(x_i) \text{ e } y_i^b = b(x_i) \text{ para } i = 0, \ldots, 2n-2.$$

- Obter pares $(x_0, q_0), \ldots, (x_{2n-2}, q_{2n-2})$
  onde $q_i = y_i^a \cdot y_i^b$ para $i = 0, \ldots, 2n-2$.

Ideia do algoritmo $\mathrm{O}(n \lg n)$

Entrada: $a = (a_0, \ldots, a_{n-1})$ e $b = (b_0, \ldots, b_{n-1})$.

- Obter pares

  $$(x_0, y_0^a), \ldots, (x_{2n-2}, y_{2n-2}^a) \text{ e } (x_0, y_0^b), \ldots, (x_{2n-2}, y_{2n-2}^b)$$

  onde $x_i \neq x_j$ para $i \neq j$, e

  $$y_i^a = a(x_i) \text{ e } y_i^b = b(x_i) \text{ para } i = 0, \ldots, 2n-2.$$

- Obter pares $(x_0, q_0), \ldots, (x_{2n-2}, q_{2n-2})$
  onde $q_i = y_i^a \cdot y_i^b$ para $i = 0, \ldots, 2n-2$.

- Determinar $q(x)$ tal que $q(x_i) = q_i$ para $i = 0, \ldots, 2n-2$.

Ideia do algoritmo $\mathrm{O}(n \lg n)$

Entrada: $a = (a_0, \ldots, a_{n-1})$ e $b = (b_0, \ldots, b_{n-1})$.

- Obter pares

  $$(x_0, y_0^a), \ldots, (x_{2n-2}, y_{2n-2}^a) \text{ e } (x_0, y_0^b), \ldots, (x_{2n-2}, y_{2n-2}^b)$$

  onde $x_i \neq x_j$ para $i \neq j$, e

  $$y_i^a = a(x_i) \text{ e } y_i^b = b(x_i) \text{ para } i = 0, \ldots, 2n-2.$$

- Obter pares $(x_0, q_0), \ldots, (x_{2n-2}, q_{2n-2})$
  onde $q_i = y_i^a \cdot y_i^b$ para $i = 0, \ldots, 2n-2$.

- Determinar $q(x)$ tal que $q(x_i) = q_i$ para $i = 0, \ldots, 2n-2$.

O passo do meio consome tempo $\mathrm{O}(n)$ trivialmente.

Ideia do algoritmo $\mathrm{O}(n \lg n)$

Dado $a = (a_0, \ldots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $\mathrm{O}(n \lg n)$?

Ideia do algoritmo $\mathrm{O}(n \lg n)$

Dado $a = (a_0, \ldots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $\mathrm{O}(n \lg n)$?

Dado x, quanto tempo levamos para calcular $a(x)$?

Ideia do algoritmo $\mathrm{O}(n \lg n)$

Dado $a = (a_0, \ldots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $\mathrm{O}(n \lg n)$?

Dado x, quanto tempo levamos para calcular $a(x)$? $\Theta(n)$.

Ideia do algoritmo $\mathrm{O}(n \lg n)$

Dado $a = (a_0, \ldots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $\mathrm{O}(n \lg n)$?

Dado x, quanto tempo levamos para calcular $a(x)$? $\Theta(n)$.

Então se escolhermos $x_0, \ldots, x_{2n-2}$ arbitrariamente,
levaremos tempo $\Theta(n^2)$ nesta etapa.

Ideia do algoritmo $\mathrm{O}(n \lg n)$

Dado $a = (a_0, \ldots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $\mathrm{O}(n \lg n)$?

Dado x, quanto tempo levamos para calcular $a(x)$? $\Theta(n)$.

Então se escolhermos $x_0, \ldots, x_{2n-2}$ arbitrariamente,
levaremos tempo $\Theta(n^2)$ nesta etapa.

Que pontos escolher então?

Ideia do algoritmo $\mathrm{O}(n \lg n)$

Dado $a = (a_0, \ldots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $\mathrm{O}(n \lg n)$?

Dado x, quanto tempo levamos para calcular $a(x)$? $\Theta(n)$.

Então se escolhermos $x_0, \ldots, x_{2n-2}$ arbitrariamente,
levaremos tempo $\Theta(n^2)$ nesta etapa.

Que pontos escolher então?

Pontos muito especiais... as raízes da unidade!

Ideia do algoritmo $\mathrm{O}(n \lg n)$

Dado $a = (a_0, \ldots, a_{n-1})$,
como calcular o valor de a em $2n - 1$ pontos em $\mathrm{O}(n \lg n)$?

Dado x, quanto tempo levamos para calcular $a(x)$? $\Theta(n)$.

Então se escolhermos $x_0, \ldots, x_{2n-2}$ arbitrariamente,
levaremos tempo $\Theta(n^2)$ nesta etapa.

Que pontos escolher então?

Pontos muito especiais... as raízes da unidade!

Quem são estas??

Raízes da unidade

São definidas para cada n.

Raízes da unidade

São definidas para cada n.

As raízes n-ésimas da unidade são as raízes da equação

$$x^n = 1.$$

(São números complexos! Lembra deles?)

Raízes da unidade

São definidas para cada n.

As raízes n-ésimas da unidade são as raízes da equação

$$x^n = 1.$$

(São números complexos! Lembra deles?)

Existem exatamente n raízes da unidade.

Raízes da unidade

São definidas para cada n.

As raízes n-ésimas da unidade são as raízes da equação

$$x^n = 1.$$

(São números complexos! Lembra deles?)

Existem exatamente n raízes da unidade.

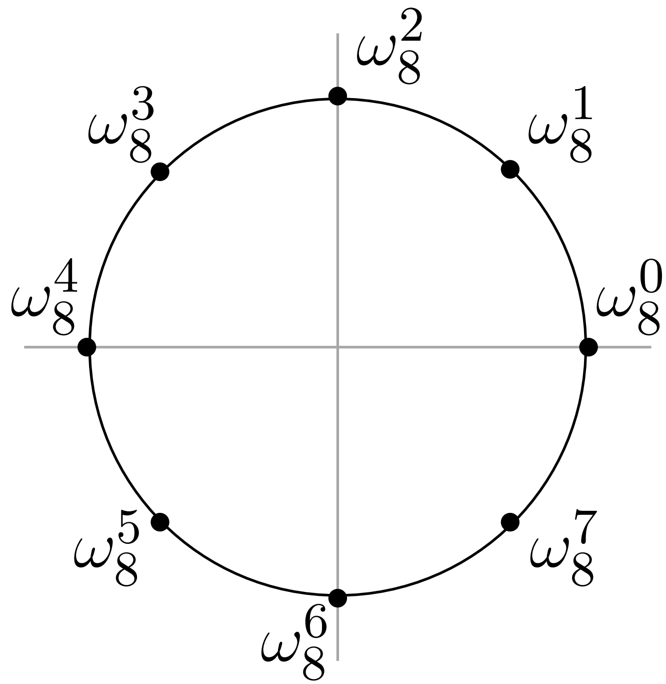
Seja $\omega_n = e^{2\pi i/n} = \cos(2\pi/n) + i\,\mathrm{sen}(2\pi/n)$.

Raízes n-ésimas da unidade: para $k = 0, 1, \ldots, n-1$,

$$\omega_n^k = e^{2\pi k i/n}.$$

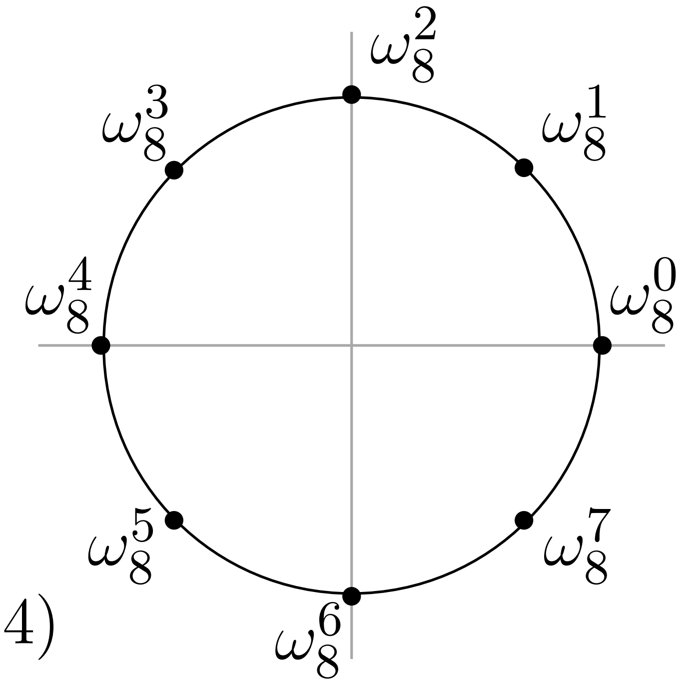
Raízes da unidade

Para $n = 8$ temos



Raízes da unidade

Para $n = 8$ temos



$$
\begin{array}{lllll}
\omega_8^0 & = & e^{0} & = & 1 \\
\omega_8^1 & = & e^{\pi i/4} & = & \cos(\pi/4) + i\,\mathrm{sen}(\pi/4) \\
\omega_8^2 & = & e^{\pi i/2} & = & \cos(\pi/2) + i\,\mathrm{sen}(\pi/2) \ = \ i \\
\omega_8^3 & = & e^{3\pi i/4} & = & \cos(3\pi/4) + i\,\mathrm{sen}(3\pi/4) \\
\omega_8^4 & = & e^{\pi i} & = & \cos(\pi) + i\,\mathrm{sen}(\pi) \ = \ -1 \\
\omega_8^5 & = & e^{5\pi i/4} & = & \cos(5\pi/4) + i\,\mathrm{sen}(5\pi/4) \\
\omega_8^6 & = & e^{3\pi i/2} & = & \cos(3\pi/2) + i\,\mathrm{sen}(3\pi/2) \ = \ -i \\
\omega_8^7 & = & e^{7\pi i/4} & = & \cos(7\pi/4) + i\,\mathrm{sen}(7\pi/4)
\end{array}
$$

Transformada discreta de Fourier

Seja $\omega_n = e^{2\pi i/n}$.

Raízes n-ésimas da unidade: para $k = 0, 1, \ldots, n-1$,

$$\omega_n^k = e^{2\pi k i/n}.$$

Dado um vetor $a = (a_0, a_1, \ldots, a_{n-1})$, representando os coeficientes de um polinômio que denotamos por $a(x)$, a transformada discreta de Fourier (DFT) de ordem n de a é o vetor $y = (y_0, y_1, \ldots, y_{n-1})$ onde $y_k = a(\omega_n^k)$ para $k = 0, 1, \ldots, n-1$.

Objetivo: programa que, dado um vetor $a = (a_0, \ldots, a_{n-1})$, determina a sua DFT de ordem n em tempo $\Theta(n \lg n)$.

Voltando ao produto de polinômios...

Lembre-se... queremos...

- Obter pares
  $$(x_0, y_0^a), \ldots, (x_{2n-2}, y_{2n-2}^a) \text{ e } (x_0, y_0^b), \ldots, (x_{2n-2}, y_{2n-2}^b)$$
  onde $x_i \neq x_j$ para $i \neq j$, e
  $$y_i^a = a(x_i) \text{ e } y_i^b = b(x_i) \text{ para } i = 0, \ldots, 2n-2.$$
- Obter pares $(x_0, q_0), \ldots, (x_{2n-2}, q_{2n-2})$
  onde $q_i = y_i^a \cdot y_i^b$ para $i = 0, \ldots, 2n-2$.
- Determinar $q(x)$ tal que $q(x_i) = q_i$ para $i = 0, \ldots, 2n-2$.

Voltando ao produto de polinômios...

Lembre-se... queremos...

- Obter pares

  $$(x_0, y_0^a), \ldots, (x_{2n-2}, y_{2n-2}^a) \text{ e } (x_0, y_0^b), \ldots, (x_{2n-2}, y_{2n-2}^b)$$

  onde $x_i \neq x_j$ para $i \neq j$, e

  $$y_i^a = a(x_i) \text{ e } y_i^b = b(x_i) \text{ para } i = 0, \ldots, 2n-2.$$

- Obter pares $(x_0, q_0), \ldots, (x_{2n-2}, q_{2n-2})$

  onde $q_i = y_i^a \cdot y_i^b$ para $i = 0, \ldots, 2n-2$.

- Determinar $q(x)$ tal que $q(x_i) = q_i$ para $i = 0, \ldots, 2n-2$.

Primeira etapa:

dados vetores $a = (a_0, \ldots, a_{n-1})$ e $b = (b_0, \ldots, b_{n-1})$, estendemos tais vetores adicionando n zeros em cada um, obtendo $a = (a_0, \ldots, a_{2n-1})$ e $b = (b_0, \ldots, b_{2n-1})$, e calculamos $\text{DFT}_{2n}(a)$ e $\text{DFT}_{2n}(b)$.

Transformada rápida de Fourier

Como calcular a $\text{DFT}_{2n}(a)$ em tempo $\mathrm{O}(n \lg n)$?

Transformada rápida de Fourier

Como calcular a DFT$_{2n}(a)$ em tempo $\mathrm{O}(n \lg n)$?

Por divisão e conquista!

Transformada rápida de Fourier

Como calcular a DFT$_{2n}(a)$ em tempo $\mathrm{O}(n \lg n)$?

Por divisão e conquista!

Considere $a^0(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{2n-2} x^{n-1}$ e
$a^1(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{2n-1} x^{n-1}$.

Transformada rápida de Fourier

Como calcular a DFT$_{2n}(a)$ em tempo $\mathrm{O}(n \lg n)$?

Por divisão e conquista!

Considere $a^0(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{2n-2} x^{n-1}$ e
$a^1(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{2n-1} x^{n-1}$.

Observe que $a(x) = a^0(x^2) + x a^1(x^2)$.

Transformada rápida de Fourier

Como calcular a $\text{DFT}_{2n}(a)$ em tempo $\mathrm{O}(n \lg n)$?

Por divisão e conquista!

Considere $a^0(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{2n-2} x^{n-1}$ e
$a^1(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{2n-1} x^{n-1}$.

Observe que $a(x) = a^0(x^2) + x a^1(x^2)$.

Com isso, calcular $a(\omega_{2n}^k)$ para $k = 0, \ldots, 2n-1$
reduz-se a calcular a^0 e a^1 em $(\omega_{2n}^k)^2$ para $k = 0, \ldots, 2n-1$.

Transformada rápida de Fourier

Como calcular a $\mathrm{DFT}_{2n}(a)$ em tempo $\mathrm{O}(n \lg n)$?

Por divisão e conquista!

Considere $a^0(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{2n-2} x^{n-1}$ e

$a^1(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{2n-1} x^{n-1}$.

Observe que $a(x) = a^0(x^2) + x a^1(x^2)$.

Com isso, calcular $a(\omega_{2n}^k)$ para $k = 0, \ldots, 2n-1$
reduz-se a calcular a^0 e a^1 em $(\omega_{2n}^k)^2$ para $k = 0, \ldots, 2n-1$.

Beleza das raízes da unidade: os quadrados das raízes de ordem $2n$ são exatamente as raízes de ordem n!
Cada raiz de ordem n aparece duas vezes.

Raízes da unidade

Propriedades:

- $$(\omega_{2n}^{k+n})^2 \;=\; \omega_{2n}^{2k} \cdot \omega_{2n}^{2n} \;=\; \omega_{2n}^{2k} \;=\; (\omega_{2n}^{k})^2$$

Raízes da unidade

Propriedades:

- $(\omega_{2n}^{k+n})^2 = \omega_{2n}^{2k} \cdot \omega_{2n}^{2n} = \omega_{2n}^{2k} = (\omega_{2n}^{k})^2$
- $\omega_{2n}^{2k} = e^{2\pi i(2k)/(2n)} = e^{2\pi i k/n} = \omega_{n}^{k}$

Raízes da unidade

Propriedades:

- $(\omega_{2n}^{k+n})^2 = \omega_{2n}^{2k} \cdot \omega_{2n}^{2n} = \omega_{2n}^{2k} = (\omega_{2n}^{k})^2$
- $\omega_{2n}^{2k} = e^{2\pi i(2k)/(2n)} = e^{2\pi i k/n} = \omega_{n}^{k}$
- $\omega_{2n}^{k+n} = -\omega_{2n}^{k}$

Raízes da unidade

Propriedades:

- $(\omega_{2n}^{k+n})^2 \;=\; \omega_{2n}^{2k} \cdot \omega_{2n}^{2n} \;=\; \omega_{2n}^{2k} \;=\; (\omega_{2n}^{k})^2$
- $\omega_{2n}^{2k} \;=\; e^{2\pi i(2k)/(2n)} \;=\; e^{2\pi ik/n} \;=\; \omega_{n}^{k}$
- $\omega_{2n}^{k+n} = -\omega_{2n}^{k}$

Algoritmo de divisão e conquista para calcular DFT$_{2n}(a)$:

Divisão: Dado $a = (a_0, \ldots, a_{2n-1})$, calcular a^0 e a^1.

Raízes da unidade

Propriedades:

- $(\omega_{2n}^{k+n})^2 = \omega_{2n}^{2k} \cdot \omega_{2n}^{2n} = \omega_{2n}^{2k} = (\omega_{2n}^{k})^2$
- $\omega_{2n}^{2k} = e^{2\pi i(2k)/(2n)} = e^{2\pi ik/n} = \omega_n^k$
- $\omega_{2n}^{k+n} = -\omega_{2n}^{k}$

Algoritmo de divisão e conquista para calcular DFT$_{2n}(a)$:

Divisão: Dado $a = (a_0, \ldots, a_{2n-1})$, calcular a^0 e a^1.

Conquista: Recursivamente calcular $y^0 =$ DFT$_n(a^0)$ e $y^1 =$ DFT$_n(a^1)$.

Raízes da unidade

Propriedades:

- $(\omega_{2n}^{k+n})^2 = \omega_{2n}^{2k} \cdot \omega_{2n}^{2n} = \omega_{2n}^{2k} = (\omega_{2n}^{k})^2$
- $\omega_{2n}^{2k} = e^{2\pi i(2k)/(2n)} = e^{2\pi ik/n} = \omega_{n}^{k}$
- $\omega_{2n}^{k+n} = -\omega_{2n}^{k}$

Algoritmo de divisão e conquista para calcular DFT$_{2n}(a)$:

Divisão: Dado $a = (a_0, \ldots, a_{2n-1})$, calcular a^0 e a^1.

Conquista: Recursivamente calcular $y^0 =$ DFT$_n(a^0)$ e $y^1 =$ DFT$_n(a^1)$.

Combinação: Para $k = 0, \ldots, n-1$, calcular $y_k = y_k^0 + \omega_{2n}^{k} y_k^1$, e para $k = n, \ldots, 2n-1$, calcular $y_k = y_{k-n}^0 + \omega_{2n}^{k} y_{k-n}^1$.

Transformada rápida de Fourier

DFT (a, n) $\triangleright$ n é uma potência de 2

se $n = 1$
    então devolva a
$a^0 \leftarrow (a_0, a_2, \ldots, a_{n-2})$
$a^1 \leftarrow (a_1, a_3, \ldots, a_{n-1})$
$y^0 \leftarrow$ DFT $(a^0, n/2)$
$y^1 \leftarrow$ DFT $(a^1, n/2)$
$\omega_n \leftarrow e^{2\pi i/n}$
$\omega \leftarrow 1$
para $k \leftarrow 0$ **até** $n/2 - 1$ **faça**
    $y_k \leftarrow y_k^0 + \omega y_k^1$
    $y_{k+n/2} \leftarrow y_k^0 - \omega y_k^1$
    $\omega \leftarrow \omega\, \omega_n$
devolva y

Consumo de tempo

O consumo de tempo do algoritmo é dado pela recorrência

$$T(n) = 2\,T(n/2) + \mathrm{O}(n).$$

Portanto é $\mathrm{O}(n \lg n)$.