Complexiteit

College 5

Docent: Lieuwe Vinkhuijzen

24 februari 2020

Oud nieuws

Huiswerkogpave 1:
`liacs.leidenuniv.nl/~vinkhuijzenlt/complexiteit`
Deadline: Maandag 2 maart 23:59

Vorige week

- Bereikbaarheid opgelost d.m.v. 2-CNF-SAT
- Selectie / Mediaan zoeken in een array
- Merge Sort

Deze week

1. Perfect matching oplossen met SAT
2. Beslissingsboom argument voor sorteren in $\Omega(n \log(n))$
3. Selectie/mediaan zoeken in array
4. Polynoom evalueren
5. Getallen vermenigvuldigen
6. Matrix vermenigvuldigen in $\mathcal{O}(n^{\log_2(7)})$

Perfect matching oplossen met SAT

Input: Een graaf $G = (V, E)$
Output: Een verzameling $M \subseteq E$, zodanig dat voor elke knoop $v \in V$: v raakt precies één tak in M.

(Voorbeelden op het bord)

Perfect matching oplossen met SAT

Input: Een graaf $G = (V, E)$
Output: Een verzameling $M \subseteq E$, zodanig dat voor elke knoop $v \in V$: v raakt precies één tak in M.

Perfect matching oplossen met SAT

Input: Een graaf $G = (V, E)$
Output: Een verzameling $M \subseteq E$, zodanig dat voor elke knoop $v \in V$: v raakt precies één tak in M.

Plan. We maken een CNF formule, ϕ, die *satisfiable* is dan een slechts dan als G een perfect matching heeft.

1. We maken voor elke tak $e \in E$ een variabele x_e
2. We zorgen dat de satisfying assignments "precies overeenkomen met" de perfect matchings van G
3. We maken twee formules α en β, en dan zetten we $\phi = \alpha \wedge \beta$
4. α is een formule die satisfiable is d.e.s.d.a. elke knoop met een tak verbonden is
5. β is een formule die satisfiable is d.e.s.d.a. elke knoop met minder dan twee takken verbonden is

Perfect matching oplossen met SAT: Constructie van α

Input: Een graaf $G = (V, E)$
Output: Een verzameling $M \subseteq E$, zodanig dat voor elke knoop $v \in V$: v raakt precies één tak in M.

(Voorbeeld op het bord)

$$
\begin{aligned}
\alpha = & (x_{1,2} \vee x_{1,5}) && \text{Vertex 1} \\
& \wedge (x_{1,2} \vee x_{2,3} \vee x_{2,6}) && \text{Vertex 2} \\
& \wedge (x_{2,3} \vee x_{3,4} \vee x_{3,6}) && \text{Vertex 3} \\
& \wedge (x_{3,4} \vee x_{4,5} \vee x_{4,6}) && \text{Vertex 4} \\
& \wedge (x_{1,5} \vee x_{4,5} \vee x_{5,6}) && \text{Vertex 5} \\
& \wedge (x_{2,6} \vee x_{3,6} \vee x_{4,6} \vee x_{5,6}) && \text{Vertex 6}
\end{aligned}
$$

Perfect matching oplossen met SAT: Constructie van β

Input: Een graaf $G = (V, E)$
Output: Een verzameling $M \subseteq E$, zodanig dat voor elke knoop $v \in V$: v raakt precies één tak in M.

(Voorbeeld op het bord)

$$
\begin{aligned}
\beta = &(x_{1,2} + x_{1,5} < 2) && \text{Vertex 1} \\
&\wedge (x_{1,2} + x_{2,3} + x_{2,6} < 2) && \text{Vertex 2} \\
&\wedge (x_{2,3} + x_{3,4} + x_{3,6} < 2) && \text{Vertex 3} \\
&\wedge (x_{3,4} + x_{4,5} + x_{4,6} < 2) && \text{Vertex 4} \\
&\wedge (x_{1,5} + x_{4,5} + x_{5,6} < 2) && \text{Vertex 5} \\
&\wedge (x_{2,6} + x_{3,6} + x_{4,6} + x_{5,6} < 2) && \text{Vertex 6}
\end{aligned}
$$

Perfect matching oplossen met SAT: Constructie van β

Input: Een graaf $G = (V, E)$
Output: Een verzameling $M \subseteq E$, zodanig dat voor elke knoop $v \in V$: v raakt precies één tak in M.

(Voorbeeld op het bord)
We kunnen $y_1 + y_2 + y_3 < 2$ omschrijven naar een CNF formule:

$$y_1 + y_2 + \cdots + y_k < 2 \iff \bigwedge_{i=1}^{k-1} \bigwedge_{j=i+1}^{k} (\neg y_i \vee \neg y_j)$$

Bijvoorbeeld:

$$y_1 + y_2 + y_3 \iff (\neg y_1 \vee \neg y_2) \wedge (\neg y_1 \vee \neg y_3) \wedge (\neg y_2 \vee \neg y_3) \quad (1)$$

Perfect matching oplossen met SAT: Constructie van β

Input: Een graaf $G = (V, E)$

Output: Een verzameling $M \subseteq E$, zodanig dat voor elke knoop $v \in V$: v raakt precies één tak in M.

(Voorbeeld op het bord)

$$
\begin{aligned}
\beta = &(\neg x_{1,2} \vee \neg x_{1,5}) && \text{Vertex 1}\\
&\wedge(\neg x_{1,2} \vee \neg x_{2,3}) \wedge (\neg x_{1,2} \vee \neg x_{2,6}) \wedge (\neg x_{2,3} \vee \neg x_{2,6}) && \text{Vertex 2}\\
&\wedge(\neg x_{2,3} \vee \neg x_{3,4}) \wedge (\neg x_{2,3} \vee \neg x_{3,6}) \wedge (\neg x_{3,4} \vee \neg x_{3,6}) && \text{Vertex 3}\\
&\wedge(\neg x_{3,4} \vee \neg x_{4,5}) \wedge (\neg x_{3,4} \vee \neg x_{4,6}) \wedge (\neg x_{4,5} \vee \neg x_{4,6}) && \text{Vertex 4}\\
&\wedge(\neg x_{1,5} \vee \neg x_{4,5}) \wedge (\neg x_{1,5} \vee \neg x_{5,6}) \wedge (\neg x_{4,5} \vee \neg x_{5,6}) && \text{Vertex 5}\\
&\wedge(\neg x_{2,6} \vee \neg x_{3,6}) \wedge (\neg x_{2,6} \vee \neg x_{4,6}) \wedge (\neg x_{2,6} \vee \neg x_{5,6}) && \text{Vertex 6}\\
&\wedge(\neg x_{3,6} \vee \neg x_{4,6}) \wedge (\neg x_{3,6} \vee \neg x_{5,6}) \wedge (\neg x_{4,6} \vee \neg x_{5,6})
\end{aligned}
$$

Perfect matching oplossen met SAT

Input: Een graaf $G = (V, E)$

Output: Een verzameling $M \subseteq E$, zodanig dat voor elke knoop $v \in V$: v raakt precies één tak in M.

Lemma

ϕ is satisfiable desda G een perfect matching heeft.

Bewijs.

($\Longleftarrow$). Stel dat $M \subseteq E$ een perfect matching is, bijvoorbeeld met de volgende takken:

$$M = \{(a, b), \ldots, (p, q), (s, t)\} \tag{2}$$

Dan bouwen we een satisfying assignment x:

$$x_{u,v} = \begin{cases} 1 & \text{Als } (u, v) \in M \\ 0 & \text{Als } (u, v) \notin M \end{cases} \tag{3}$$

Perfect matching oplossen met SAT

Input: Een graaf $G = (V, E)$
Output: Een verzameling $M \subseteq E$, zodanig dat voor elke knoop $v \in V$: v raakt precies één tak in M.

Lemma

ϕ is satisfiable desda G een perfect matching heeft.

Bewijs.

($\Longrightarrow$). Stel dat ϕ een satisfying assignment x is, dan is de volgende M een perfect matching.

$$M = \{(u, v) \mid x_{u,v} = 1\} \tag{4}$$

Immers, alle knopen in M hebben nu precies één buur in M. $\square$

Ondergrens: Sorteren in $\Omega(n \log n)$

Theorem

Elk algoritme dat een lijst van n getallen sorteert, en dat dat alleen middels vergelijkingen doet, doet in de worst case minstens $O(n \log n)$ vergelijkingen.

Bewijs.

Met een beslissingsboomargument. □

Ondergrens: Sorteren in $\Omega(n \log n)$

We bekijken sorteeralgoritmen gebaseerd op het doen van vergelijkingen van de vorm $A[i] < A[j]$.

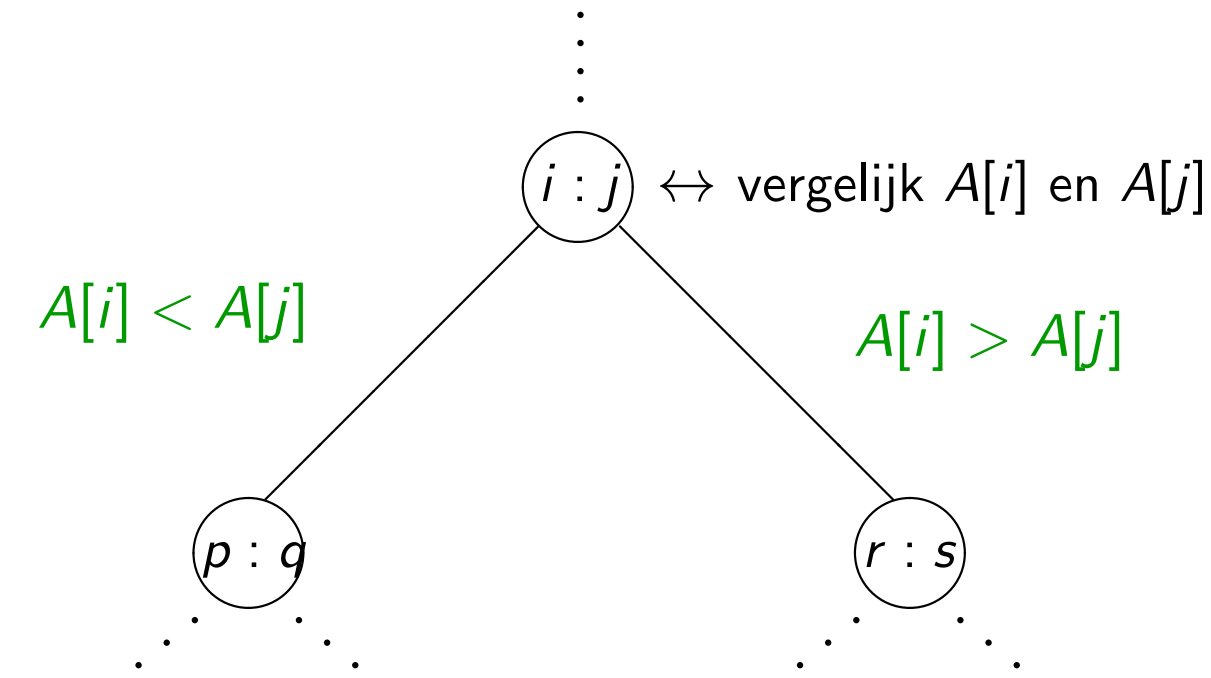
Aannames (z.b.d.a.):

- A bevat n verschillende waarden. (We gaan immers een ondergrens voor de worst case bepalen.)
- het sorteeralgoritme stopt zodra de sortering (onderlinge ordening) gevonden is.

Ondergrens: Sorteren in $\Omega(n \log n)$

Algoritme gebaseerd op array-vergelijkingen $A[i] < A[j]$	$\leftrightarrow$	Beslissingsboom
Arravergelijking	$\leftrightarrow$	Interne knoop
Gevonden volgorde	$\leftrightarrow$	Blad
Executie van algoritme	$\leftrightarrow$	Pad: wortel naar blad

Ondergrens: Sorteren in $\Omega(n \log n)$

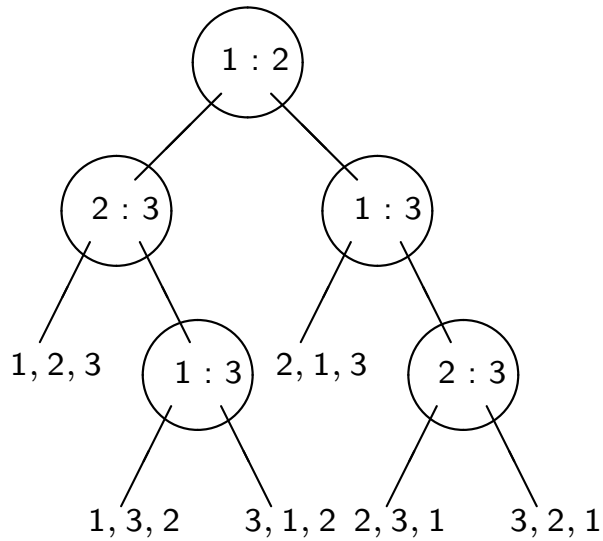


vergelijk vervolgens
$A[p]$ en $A[q]$

vergelijk vervolgens
$A[r]$ en $A[s]$

Beslissingsboom voor algoritmen gebaseerd op arrayvergelijkingen

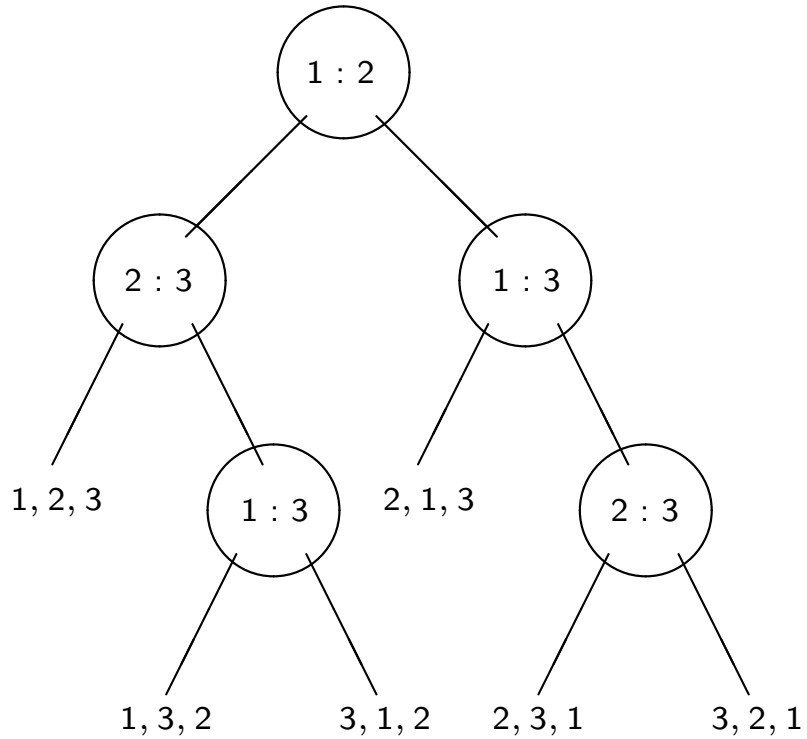
Ondergrens: Sorteren in $\Omega(n \log n)$



Beslissingsboom voor Insertion sort met $n = 3$

2, 3, 1 betekent: $A[2] < A[3] < A[1]$ (analoog de andere bladeren)

Ondergrens: Sorteren in $\Omega(n \log n)$



In een beslissingsboom voor algoritmen gebaseerd op arrayvergelijkingen geeft de hoogte van de boom precies het aantal vergelijkingen in de worst case aan.

Ondergrens: Sorteren in $\Omega(n \log n)$

1. - alleen de onderlinge volgorde van de array-elementen wordt onderscheiden; niet de waarde
   - het rijtje $6, 11, 15, 8, 3$ wordt dus precies zo behandeld door het sorteeralgoritme als het rijtje $2, 4, 5, 3, 1$
   - ze volgen dan ook precies hetzelfde pad in de beslissingsboom
   - er zijn in essentie $n!$ mogelijke te onderscheiden invoeren, die elk één pad volgen in de boom $\Rightarrow$ er zijn maximaal $n!$ bladeren

Ondergrens: Sorteren in $\Omega(n \log n)$

2.
   - sorteren = vind de oplopende ordening
   - er zijn dus $n!$ verschillende eindantwoorden (= ordeningen) mogelijk
   - een sorteeralgoritme moet die allemaal kunnen vinden
   - de bijbehorende beslissingsboom moet dus minstens $n!$ bladeren hebben
3. Conclusie: een beslissingsboom corresponderend met een sorteeralgoritme gebaseerd op arrayvergelijkingen heeft precies $n!$ bladeren (n = aantal array-elementen)

Ondergrens: Sorteren in $\Omega(n \log n)$

Stelling Het aantal vergelijkingen in de worst case is voor elk algoritme dat sorteert middels arrayvergelijkingen ten minste $\lceil \lg n! \rceil$ (dus $\Omega(n \lg n)$).

Ondergrens: Sorteren in $\Omega(n \log n)$

Gegeven een binaire boom $\mathcal{B}$ met b bladeren.
Definitie. De externe padlengte E van $\mathcal{B}$ is de som van de lengtes van alle paden van de wortel naar een blad:

$$E = \Sigma_{\text{bladeren}}(\text{lengte pad wortel} \longrightarrow \text{blad})$$

Lemma. Zij E de externe padlengte van $\mathcal{B}$. Dan geldt:

$$E \geq b \cdot (\lceil \lg b \rceil - 1)$$

Gevolg. De gemiddelde lengte van een pad van de wortel naar een blad $= \frac{E}{b} \geq \lceil \lg b \rceil - 1$.

Ondergrens: Sorteren in $\Omega(n \log n)$

Stelling Het aantal vergelijkingen in de average case is voor elk algoritme dat sorteert middels arrayvergelijkingen $\Omega(n \lg n)$.
Dit onder de aanname dat alle $n!$ mogelijke volgordes als invoerrijtje even waarschijnlijk zijn.

k-de kleinste zoeken in array

Input: Een getal k, een array getallen $A[1], \ldots, A[n]$.
Output: De $A[i]$ die groter is dan precies $k-1$ andere elementen in A. (Met andere woorden: het k-de kleinste element)

k-de kleinste zoeken in array

1: **procedure** MEDIAAN($A[1], \ldots, A[n]$)
2: Verdeel de getallen in $\lfloor \frac{n}{5} \rfloor$ groepjes van 5 elementen (en 1 groepje met de resterende n mod 5)
3: Vindt de mediaan van elk groepje
4: $m :=$ de mediaan van de $\lceil \frac{n}{5} \rceil$ medianen[1]
5: Herorganiseer het array A rond m:

$< m$	m	$> m$

6: $i_m :=$ nieuwe index van m, i.e. $A[i_m] = m$
7: **if** $i_m = \lceil \frac{n}{2} \rceil$ **then**
8: Gevonden!
9: **else if** $i_m > \frac{1}{2}n$ **then**
10: **return** SELECT($\frac{n}{2}, A[1], \ldots, A[i_m - 1]$)
11: **else**
12: **return** SELECT($\frac{n}{2} - i_m, A[i_m + 1], \ldots, A[n]$)
13: **end if**
14: **end procedure**

[1] m is niet per sé de mediaan van A

k-de kleinste zoeken in array

1: **procedure** SELECT($k, A[1], \ldots, A[n]$)
2: Verdeel de getallen in $\lfloor \frac{n}{5} \rfloor$ groepjes van 5 elementen (en 1 groepje met de resterende n mod 5)
3: Vindt de mediaan van elk groepje
4: $m :=$ de mediaan van de $\lceil \frac{n}{5} \rceil$ medianen[2]
5: Herorganiseer het array A rond m:

$< m$	m	$> m$

6: $i_m :=$ nieuwe index van m, i.e. $A[i_m] = m$
7: **if** $i_m = k$ **then**
8: Gevonden!
9: **else if** $i_m > k$ **then**
10: **return** SELECT($k, A[1], \ldots, A[i_m - 1]$)
11: **else**
12: **return** SELECT($k - i_m, A[i_m + 1], \ldots, A[n]$)
13: **end if**
14: **end procedure**

[2] m is niet per sé de mediaan van A

Selectie is $O(n)$

Laat $T(n) =$ aantal vergelijkingen in de worst case dat het (recursieve) algoritme SELECT doet.

$$T(n) = \begin{cases} 1 & \text{Als } n \text{ klein genoeg} \\ T(\lceil \frac{n}{5} \rceil) + T(\lceil \frac{7n}{10} \rceil + 6) + n & \text{Als } n \text{ groot} \end{cases}$$

Bewering:

$T(n) \leq cn$ voor geschikte c en $n \geq \cdots$

ofwel: $T(n) \in O(n)$

Polynoomevaluatie

Input: Reële getallen $x, a_0, a_1, \ldots, a_n \in \mathbb{R}$
Output: De waarde
$p(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-1} \cdot x^{n-1} + a_n \cdot x^n$

Polynoomevaluatie

Input: Reële getallen $x, a_0, a_1, \ldots, a_n \in \mathbb{R}$
Output: De waarde
$p(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-1} \cdot x^{n-1} + a_n \cdot x^n$

```
1:  procedure POLYNOOM(x, a_0, a_1, ..., a_n)
2:      p := 0
3:      for k = 0 ··· n do
4:          m := 1
5:          for i = 0 ... k do                    ▷ Berekent x^k
6:              m := m · x
7:          end for
8:          p := p + a_k m
9:      end for
10: end procedure
```

Polynoomevaluatie

Input: Reële getallen $x, a_0, a_1, \ldots, a_n \in \mathbb{R}$

Output: De waarde

$p(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-1} \cdot x^{n-1} + a_n \cdot x^n$

```
1:  procedure POLYNOOM(x, a_0, a_1, ..., a_n)
2:      p := 0
3:      for k = 0 ··· n do
4:          m := 1
5:          for i = 0 ... k do                    ▷ Berekent x^k
6:              m := m · x
7:          end for
8:          p := p + a_k m
9:      end for
10: end procedure
```

Aantal optellingen: n

Aantal vermenigvuldigingen:

$\sum_{k=0}^{n} \sum_{i=0}^{k} 1 = \sum_{k=0}^{n} k = \frac{1}{2} n(n+1) \in \Theta(n^2)$

Polynoomevaluatie

Input: Reële getallen $x, a_0, a_1, \ldots, a_n \in \mathbb{R}$
Output: De waarde
$p(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-1} \cdot x^{n-1} + a_n \cdot x^n$

```
1: procedure POLYNOOM(x, a_0, a_1, ..., a_n)
2:     p := a_0 + a_1 · x
3:     macht := x
4:     for k = 2 ... n do
5:         macht := macht · x                ▷ Berekent x^k
6:         p := p + a_k · macht
7:     end for
8:     return p
9: end procedure
```

Polynoomevaluatie

Input: Reële getallen $x, a_0, a_1, \ldots, a_n \in \mathbb{R}$
Output: De waarde
$p(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-1} \cdot x^{n-1} + a_n \cdot x^n$

```
1: procedure POLYNOOM(x, a_0, a_1, ..., a_n)
2:     p := a_0 + a_1 · x
3:     macht := x
4:     for k = 2 ... n do
5:         macht := macht · x                    ▷ Berekent x^k
6:         p := p + a_k · macht
7:     end for
8:     return p
9: end procedure
```

Aantal optellingen: n
Aantal vermenigvuldigingen: $2n - 1$

Polynoomevaluatie: Methode van Horner

Input: Reële getallen $x, a_0, a_1, \ldots, a_n \in \mathbb{R}$
Output: De waarde
$p(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-1} \cdot x^{n-1} + a_n \cdot x^n$

Observatie:

$$
\begin{aligned}
a_2x^2 + a_1x + a_0 &= (a_2x + a_1)x + a_0 \\
a_3x^3 + a_2x^2 + a_1x + a_0 &= ((a_3x + a_2)x + a_1)x + a_0 \\
a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 &= (((a_4x + a_3)x + a_2)x + a_1)x + a_0
\end{aligned}
$$

Polynoomevaluatie: Methode van Horner

Input: Reële getallen $x, a_0, a_1, \ldots, a_n \in \mathbb{R}$
Output: De waarde
$p(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-1} \cdot x^{n-1} + a_n \cdot x^n$

Observatie:

$$
\begin{aligned}
a_2x^2 + a_1x + a_0 &= (a_2x + a_1)x + a_0 \\
a_3x^3 + a_2x^2 + a_1x + a_0 &= ((a_3x + a_2)x + a_1)x + a_0 \\
a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 &= (((a_4x + a_3)x + a_2)x + a_1)x + a_0
\end{aligned}
$$

Polynoomevaluatie: Methode van Horner

Input: Reële getallen $x, a_0, a_1, \ldots, a_n \in \mathbb{R}$
Output: De waarde
$p(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-1} \cdot x^{n-1} + a_n \cdot x^n$

```
1: procedure POLYNOOM(x, a_0, a_1, ..., a_n)
2:     p := a_n
3:     macht := x
4:     for k = n − 1 down to 0 do
5:         p := p · x + a_k
6:     end for
7:     return p
8: end procedure
```

Aantal optellingen: n
Aantal vermenigvuldigingen: n ☺

Getallen vermenigvuldigen in $O(n^2)$

Input: Twee gehele getallen a, b van n bits
Output: Het getal $c = ab$

Getallen vermenigvuldigen in $O(n^2)$

Input: Twee gehele getallen a, b van n bits
Output: Het getal $c = ab$

$$a = a_0 + 10a_1 + 10^2 a_2 + 10^3 a_3 + \cdots 10^{n-1} a_{n-1}$$
$$b = b_0 + 10b_1 + 10^2 b_2 + 10^3 b_3 + \cdots 10^{n-1} b_{n-1}$$

1: **procedure** VERMENIGVULDIG(a, b)
2: **for** $i = 0 \cdots n-1$ **do**
3: **for** $j = 0 \ldots n-1$ **do**
4: Bereken $a_i \times b_j$
5: **end for**
6: **end for**
7: **return** $\sum_{k=0}^{2n} 10^k \left(\sum_{i+j=k} a_i \times b_j \right)$
8: **end procedure**

Getallen vermenigvuldigen in $O(n^{\log_2(3)})$

Karatsuba's algoritme
Tijd: $T(n) \in O(n^{\log_2(3)}) = O(n^{1.58})$
(Op het bord)

Matrix vermenigvuldiging in $O(n^{\log_2(7)})$

Input: Twee $n \times n$ matrices van gehele getallen A, B
Output: Het product $C = A \cdot B$

Matrix vermenigvuldiging in $O(n^3)$

Input: Twee $n \times n$ matrices van gehele getallen A, B
Output: Het product $C = A \cdot B$

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{21} & A_{2,2} \end{bmatrix} B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{21} & B_{2,2} \end{bmatrix}$$

$$C = A \cdot B = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

Matrix vermenigvuldiging in $O(n^3)$

Input: Twee $n \times n$ matrices van gehele getallen A, B
Output: Het product $C = A \cdot B$

```
1:  procedure MatMul(A, B)
2:      for i = 1 ... n do
3:          for j = 1 ... n do
4:              C_{i,j} := 0
5:              for k = 1 ... n do
6:                  C_{i,j} := C_{i,j} + A_{i,k} × B_{k,j}
7:              end for
8:          end for
9:      end for
10:     return C
11: end procedure
```

Matrix vermenigvuldiging in $O(n^3)$

Input: Twee $n \times n$ matrices van gehele getallen A, B
Output: Het product $C = A \cdot B$

Plan: Verdeel en heers. Neem het geval dat A een $2^k \times 2^k$ matrices zijn.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} B = \begin{bmatrix} B_{1,1} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C = A \cdot B = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

Aantal optellingen: 4
Aantal vermenivuldigingen: 8

Matrix vermenigvuldiging in $O(n^3)$

Input: Twee $n \times n$ matrices van gehele getallen A, B
Output: Het product $C = A \cdot B$

Aantal optellingen:

$$A(n) = \begin{cases} 0 & \text{Als } n = 1 \\ 8A(\frac{1}{2}n) + n^2 & \text{Als } n > 1 \end{cases} = \frac{4}{3}n^2 \qquad (5)$$

Aantal vermenigvuldigingen:

$$M(n) = \begin{cases} 8 & \text{Als } n = 2 \\ 8M(\frac{1}{2}n) & \text{Als } n > 2 \end{cases} = 8^{\log_2(n)} = 3^n \qquad (6)$$

Matrix vermenigvuldiging in $O(n^3)$

Input: Twee $n \times n$ matrices van gehele getallen A, B
Output: Het product $C = A \cdot B$

Aantal optellingen:

$$A(n) = \begin{cases} 0 & \text{Als } n = 1 \\ 8A(\frac{1}{2}n) + n^2 & \text{Als } n > 1 \end{cases} = \frac{4}{3}n^2 \qquad (7)$$

Aantal vermenigvuldigingen:

$$M(n) = \begin{cases} 8 & \text{Als } n = 2 \\ 8M(\frac{1}{2}n) & \text{Als } n > 2 \end{cases} = 8^{\log_2(n)} = 3^n \qquad (8)$$