Zevende college Complexiteit

21 maart 2025

Quicksort, Shellsort

- Mergesort
  - verdeel-en-heers: sorteert *recursief* twee helften en voegt ze samen tot één gesorteerde rij
  - best case, worst case, average case $\in \Theta(n \lg n)$
- Sorteren met behulp van arrayvergelijkingen: zowel worst case als average case $\in \Omega(n \lg n)$

Mergesort en Quicksort zijn sorteermethoden die allebei zijn gebaseerd op de verdeel-en-heers strategie:

```
sorteer(rij)::
    if ( de rij heeft meer dan één element ) then
        verdeel de rij in twee stukken: linkerrij en rechterrij;
        sorteer(linkerrij);
        sorteer(rechterrij);
        combineer linkerrij en rechterrij;
    fi .
```

Mergesort stopt het meeste werk in de combineerstap, Quicksort in de verdeelstap. Beide sorteermethoden zijn gebaseerd op arrayvergelijkingen, en verwisselen array-elementen die i.h.a. verder van elkaar liggen.

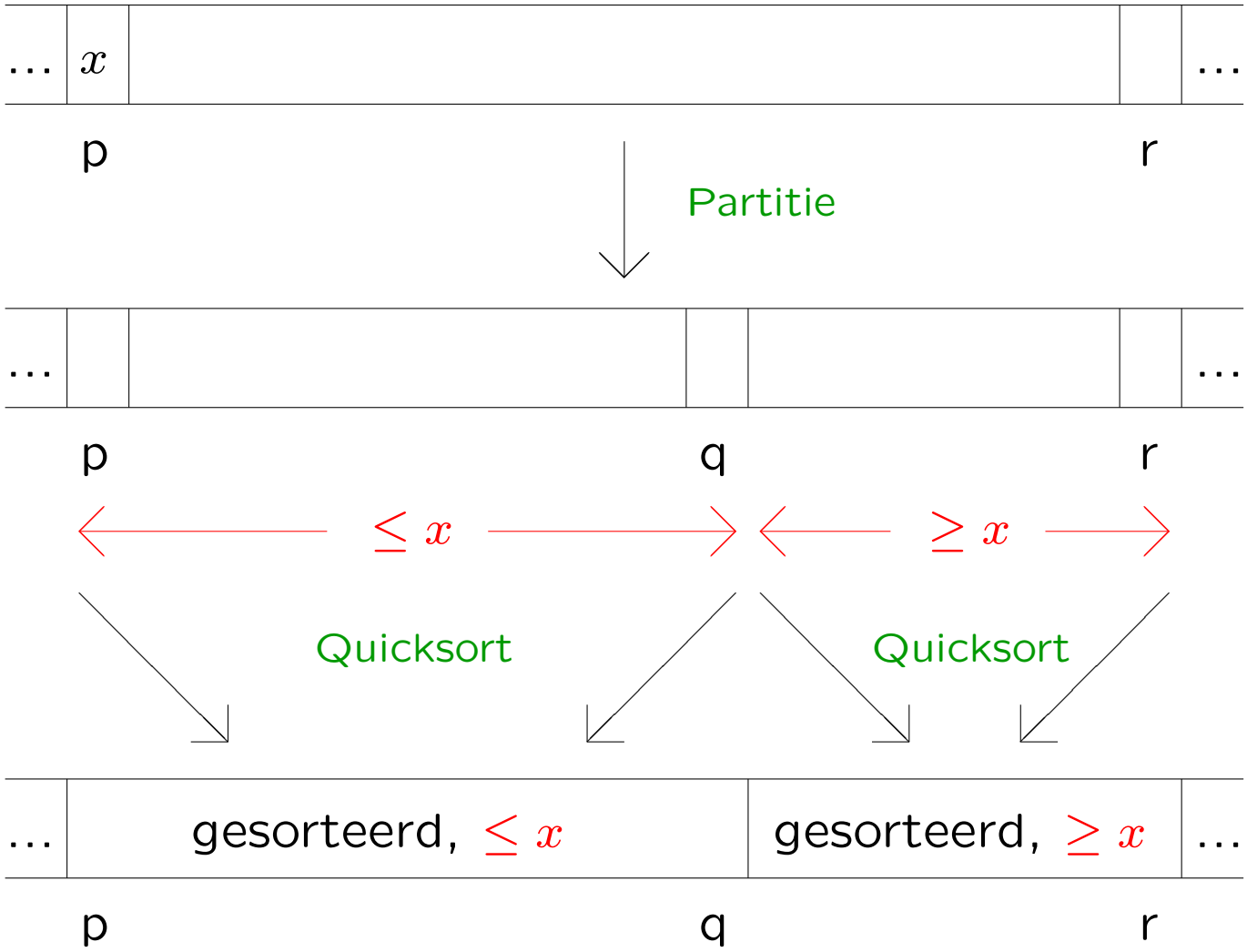
Quicksort*

```
QuickSort(A, p, r)::
// sorteert A[p], ..., A[r] oplopend
        if p < r then
                q := Partitie(A, p, r);
                QuickSort(A, p, q);
                QuickSort(A, q + 1, r);
        fi
```

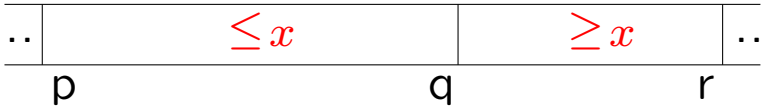
Aanroep:
Quicksort(A,1,n)

- recursief
- alleen interne verwisselingen
- geen extra geheugenruimte
- in de praktijk een van de snelste

*Tony Hoare, 1962; Turing Award 1980


...
x
...
p
r
Partitie
...
...
p
q
r
$\leq x$
$\geq x$
Quicksort
Quicksort
...
gesorteerd, $\leq x$
gesorteerd, $\geq x$
...
p
q
r

Partitie (A, p, r)::
// reorganiseert het (deel)array $A[p], \ldots, A[r]$ als volgt:



(*) // hier komt nog wat
$x = A[p]$; $i := p - 1$; $j := r + 1$;
while $i < j$ **do**
    $j := j - 1$; // loop met j naar links
    while $A[j] > x$ **do**
        $j := j - 1$;
    od // tot je een waarde $A[j] \leq x$ vindt
    $i := i + 1$; // loop met i naar rechts
    while $A[i] < x$ **do**
        $i := i + 1$;
    od // tot je een waarde $A[i] \geq x$ vindt
    if $i < j$ **then**
        wissel$(A[i], A[j])$;
    fi // $A[i]$ en $A[j]$ staan nu in het goede stuk
od
return j; // dit wordt dus q

- de *basisoperatie* is het vergelijken van array-elementen: $A[j] > x$ en $A[i] < x$

- in ronde 1 stopt i op positie p en j op een positie $\geq p$

- na ronde 1 is $A[p] \leq x$, dus j zal nooit het array uitlopen

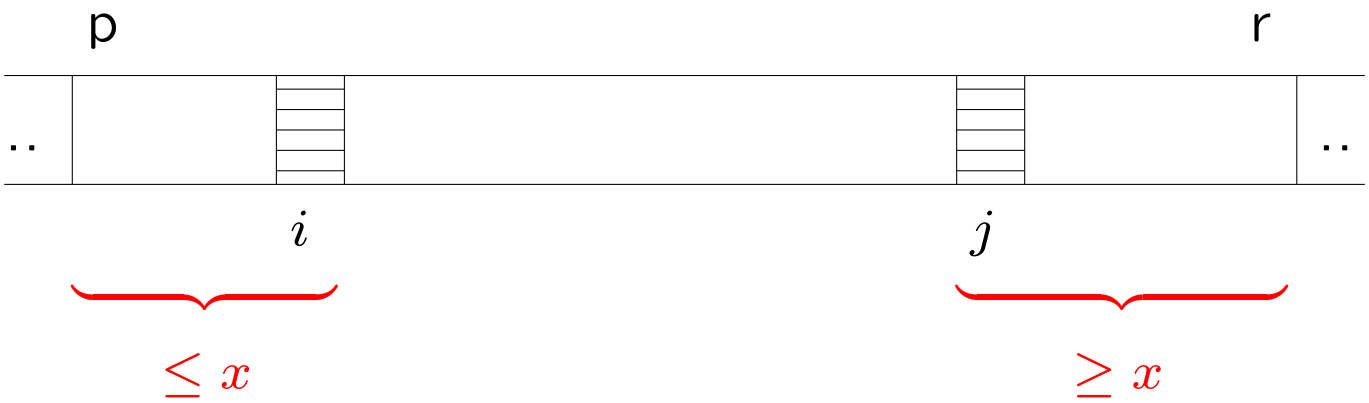
- na ronde 1 is ofwel $i = j = p$ (en dan zijn we klaar), ofwel $p < j \leq r$ en dan zal j daarna $\leq r - 1$ worden

- na afloop is altijd $p \leq j \leq r - 1$, dus $p \leq q \leq r - 1$

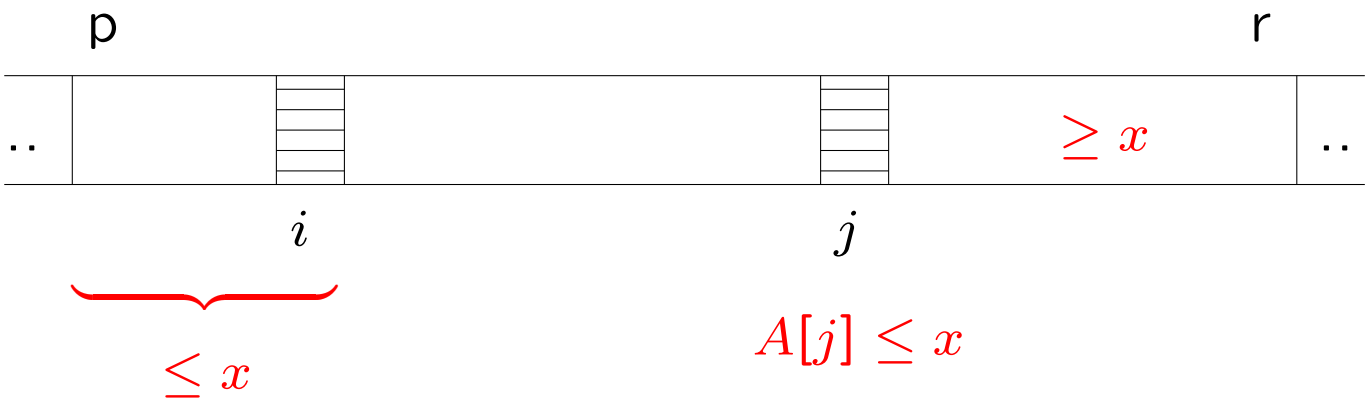
- Partitie stopt uiteindelijk met $i = j$ of $i = j+1$, dus $A[q] \leq x$

- elk array-element wordt precies één keer vergeleken met x, behalve $A[q]$ (twee keer) en $A[q + 1]$ (soms twee keer)

1. Na een volledige ronde



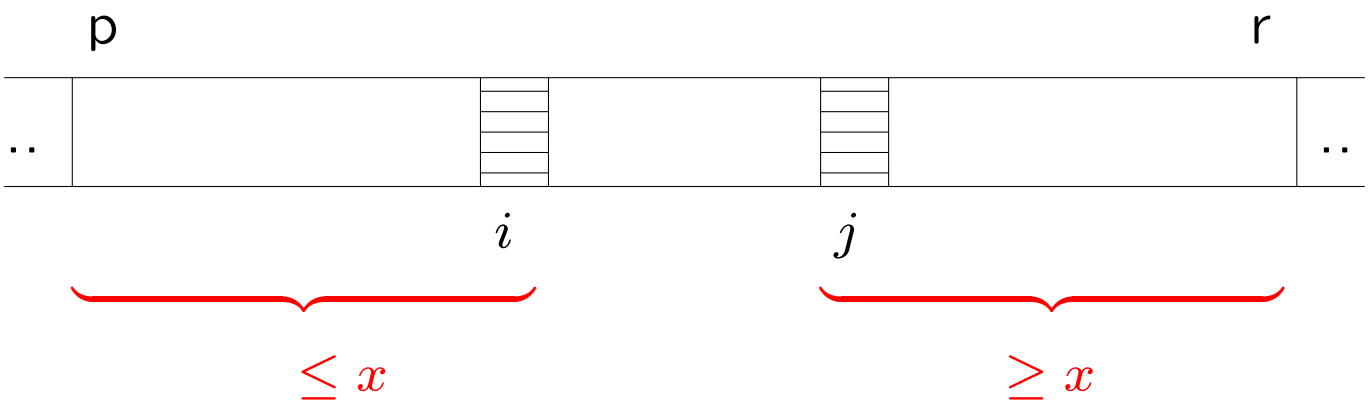
2. Na de j-loop (j stopt altijd $\geq i$)



3. Na de i-loop, vóór de verwisseling (i stopt altijd $\leq j+1$)

$A[i] \geq x$ $A[j] \leq x$

4. Na de verwisseling



1. Na afloop van Partitie(A, p, r) is altijd $p \leq q \leq r - 1$. Quicksort wordt dus op echt kleinere rijtjes recursief aangeroepen.

2. Partitie doet altijd $\Theta(m)$ vergelijkingen, nl. $m+1$ of $m+2$, met m het aantal elementen van het (deel)array $A[p], \ldots, A[r]$

3. Er worden elementen verwisseld die ver uit elkaar kunnen liggen. Per vergelijking worden dus wellicht > 1 inversies opgeheven.

1. Wat gebeurt er met gelijke array-elementen? Bijvoorbeeld: $5, 3, 5, 5, 6, 5, 7$ of $5, 6, 3, 4, 3, 8, 3$ of een array bestaande uit allemaal dezelfde elementen.

2. Wat gebeurt er met een rijtje met allemaal verschillende elementen? Bijvoorbeeld: $3, 2, 7, 4, 6, 8, 5, 1$ of een reeds gesorteerd rijtje of een omgekeerd gesorteerd rijtje.

3. Zie ook opgave 40 en 41 uit het dictaat.

1. Bad case voor Quicksort:

   Op een reeds gesorteerd rijtje (!) bestaande uit verschillende waardes, bijvoorbeeld $1, 2, \ldots, n$, doet Quicksort

$$\sum_{k=3}^{n+1} k = \frac{1}{2} n(n+3) - 2 \in \Theta(n^2)$$

   arrayvergelijkingen. Partitie deelt het array telkens zeer onevenwichtig in tweeën.

2. Good case voor Quicksort (n een tweemacht):

$$B(n) = \begin{cases} 0 & n = 1 \\ 3 & n = 2 \\ 2\,B(\frac{n}{2}) + \Theta(n) & n = 2^k, n > 2 \end{cases}$$

Het aantal arrayvergelijkingen $B(n)$ is dan $\Theta(n \lg n)$. Dit komt voor als Partitie het array bij elke aanroep precies in tweeën deelt. Dit is zelfs het *beste* geval voor Quicksort.

Laat $W(n)$ het aantal arrayvergelijkingen voorstellen dat Quicksort doet in de worst case. Dan voldoet W aan*:

$$W(n) = \begin{cases} 0 & n = 1 \\ \max_{1 \leq q \leq n-1}(W(q) + W(n-q)) + \Theta(n) & n > 1 \end{cases}$$

Er geldt dat $W(n) \leq dn^2$ voor zekere $d > 0$, en vanaf zekere n_0. Dus: $W(n) \in O(n^2)$.

Samen met de bad case hebben we dan:

Stelling
Quicksort doet $\Theta(n^2)$ arrayvergelijkingen in de worst case.

*i.p.v. $\Theta(n)$ kun je in de recurrente betrekking ook $\leq n + 2$ zetten

De keuze van de spil (x dus) heeft grote invloed op de complexiteit van Quicksort. Standaard het eerste array-element ($A[p]$) kiezen als spil is een slechte keuze.

Voeg ter verbetering op plek (*) in Partitie toe:

Kies een slim array-element en wissel dat met $A[p]$.

Slim kan zijn: kies een willekeurig* (random) array-element. De worst case blijft dan uiteraard $\Theta(n^2)$, maar onder de aanname dat alle $A[i]$ verschillend zijn hebben we nu:

Stelling
Quicksort doet $O(n \lg n)$ arrayvergelijkingen in de average case.

*Randomised Quicksort

Partitie (A, p, r)::
// reorganiseert het (deel)array $A[p], \ldots, A[r]$ als volgt:

kies *random* array-element en wissel met $A[p]$.
$x = A[p]$; $i := p - 1$; $j := r + 1$;
while $i < j$ **do**
    $j := j - 1$; // loop met j naar links
    while $A[j] > x$ **do**
        $j := j - 1$;
    od // tot je een waarde $A[j] \leq x$ vindt
    i := i + 1; // loop met i naar rechts
    while $A[i] < x$ **do**
        $i := i + 1$;
    od // tot je een waarde $A[i] \geq x$ vindt
    if $i < j$ **then**
        wissel$(A[i], A[j])$;
    fi // $A[i]$ en $A[j]$ staan nu weer in het goede stuk
od
return j; // dit wordt dus q

Algoritme	worst case	average case	extra ruimte
Insertion sort	$\frac{1}{2}n^2$	$\Theta(n^2)$	in situ (d.w.z. geen)
Quicksort	$\frac{1}{2}n^2$	$\Theta(n \lg n)$	evenredig met $\lg n$
Mergesort	$n \lg n$	$\Theta(n \lg n)$	evenredig met n
Heapsort	$2n \lg n$	$\Theta(n \lg n)$	in situ

Heapsort: zie dictaat, opgave 38

Vergelijking van verschillende sorteeralgoritmen

(average case; tijd in seconden)

n	Insertion sort $O(n^2)$	Shellsort $O(n^{7/6})$	Heapsort $O(n \lg n)$	Quicksort $O(n \lg n)$	Quicksort* $O(n \lg n)$
10	0,00044	0,00041	0,00057	0,00052	0,00046
100	0,00675	0,00171	0,00420	0,00284	0,00244
1000	0,59564	0,02927	0,05565	0,03153	0,02587
10.000	58,864	0,42998	0,71650	0,36765	0,31532
100.000	—	5,7298	8,8591	4,2298	3,5882
1.000.000	—	71,164	104,68	47,065	41,282

*geoptimaliseerd

Shellsort*

Shellsort was een van de eerste sorteeralgoritmen met worst case complexiteit minder dan kwadratisch.

Shellsort sorteert in elke ronde deelrijtjes. In het begin veel korte rijtjes, waarbij de elementen uit een rijtje ver van elkaar liggen. Later weinig lange rijtjes, waarbij de elementen uit een rijtje dicht bij elkaar liggen. De rij wordt zo als het ware voorgesorteerd. In de laatste ronde wordt de rij dan als geheel gesorteerd. Shellsort sorteert met behulp van vergelijk-verwissel (compare-exchange, indien nodig) operaties.

*Donald Shell, 1959

Definitie

Een rij $A[1], A[2], \ldots, A[n]$ heet k-gesorteerd als geldt:
$A[i] \leq A[i+k]$ voor elke $i = 1, 2, \ldots, n-k$.

Voorbeeld: Het rijtje

$$5, 8, 24, 13, 7, 18, 31, 19, 44, 63, 82, 29$$

is 4-gesorteerd (en overigens ook 6-gesorteerd).

Merk op: 1-gesorteerd = gesorteerd.

Shellsort gebruikt een rijtje stapgroottes (increments, sprongafstanden) $h_t, h_{t-1}, \ldots, h_2, h_1 = 1$. De rij A met n elementen wordt gesorteerd door achtereenvolgens deelrijen te sorteren* van elementen die telkens op afstand h_i van elkaar liggen. Met andere woorden: A wordt h_i-gesorteerd voor $i = t, \ldots, 1$.
Aangezien $h_1 = 1$ sorteert Shellsort correct.

Merk op: bij het k-sorteren worden k deelrijtjes van elk maximaal $\lceil \frac{n}{k} \rceil$ elementen gesorteerd.

*bijvoorbeeld met Insertion sort

$h_3 = 6, h_2 = 3, h_1 = 1, n = 13$

$81, 94, 11, 96, 12, 35, 17, 95, 28, 58, 41, 75, 15$

$\downarrow$ 6-sorteren

$15, 94, 11, 58, 12, 35, 17, 95, 28, 96, 41, 75, 81$

$\downarrow$ 3-sorteren

$15, 12, 11, 17, 41, 28, 58, 94, 35, 81, 95, 75, 96$

$\downarrow$ 1-sorteren

$11, 12, 15, 17, 28, 35, 41, 58, 75, 81, 94, 95, 96$

Voorbeeld

$h_3 = 6, h_2 = 3, h_1 = 1, n = 13$

81, 94, 11, 96, 12, 35, 17, 95, 28, 58, 41, 75, 15

↓ 6-sorteren

15, 94, 11, 58, 12, 35, 17, 95, 28, 96, 41, 75, 81

↓ 3-sorteren

15, 12, 11, 17, 41, 28, 58, 94, 35, 81, 95, 75, 96

↓ 1-sorteren

11, 12, 15, 17, 28, 35, 41, 58, 75, 81, 94, 95, 96

Voorbeeld

Insertion sort op het voorbeeldrijtje: 52 vergelijkingen.

Shellsort met Insertion sort: 44 vergelijkingen:

$81, 94, 11, 96, 12, 35, 17, 95, 28, 58, 41, 75, 15$

inversies = 41 $\downarrow$ 6-sorteren: 8 vergelijkingen

$15, 94, 11, 58, 12, 35, 17, 95, 28, 96, 41, 75, 81$

inversies = 25 $\downarrow$ 3-sorteren: 15 vergelijkingen

$15, 12, 11, 17, 41, 28, 58, 94, 35, 81, 95, 75, 96$

inversies = 11 $\downarrow$ 1-sorteren: 21 vergelijkingen

$11, 12, 15, 17, 28, 35, 41, 58, 75, 81, 94, 95, 96$

```
(1)   h = n div 2; // h_t = ⌊n/2⌋ dus
(2)   while h > 0 do
(3)       for i := h + 1 to n do
(4)           temp := A[i]; j := i;
(5)           while j − h > 0 do
              // invoegen op de juiste plek in je eigen deelrijtje
(6)               if temp < A[j − h] then
(7)                   A[j] := A[j − h];                    // schuif
(8)                   j := j − h;
(9)               else
(10)                  "exit binnenste while";
(11)              fi
(12)          od
(13)          A[j] := temp;                                // zet neer
(14)      od
          // de rij is nu h-gesorteerd
(15)      h := h div 2; // oorspronkelijke keuze van Shell: h_i = ⌊h_{i+1}/2⌋
(16)  od
```

Merk op: regel (4) t/m (13) is Insertion sort op deelrijtjes

Een zeer belangrijke eigenschap van Shellsort is de volgende (zonder bewijs).

Stelling

Als een ℓ-gesorteerd array h-gesorteerd wordt met behulp van vergelijk-verwissel-operaties, dan blijft het ℓ-gesorteerd.

Voorbeeld met $n = 12$ en incrementrijtje $6, 4, 3, 2, 1$:

$7, 19, 24, 13, 31, 8, 82, 18, 44, 63, 5, 29$

$\downarrow$ 6-sorteren

$7, 18, 24, 13, 5, 8, 82, 19, 44, 63, 31, 29$ 6-gesorteerd

$\downarrow$ 4-sorteren

$5, 8, 24, 13, 7, 18, 31, 19, 44, 63, 82, 29$ $\{4, 6\}$-gesorteerd

$\downarrow$ 3-sorteren

$5, 7, 18, 13, 8, 24, 31, 19, 29, 63, 82, 44$ $\{3, 4, 6\}$-gesorteerd

$\downarrow$ 2-sorteren

$5, 7, 8, 13, 18, 19, 29, 24, 31, 44, 82, 63$ $\{2, 3, 4, 6\}$-gesorteerd

$\downarrow$ 1-sorteren

$5, 7, 8, 13, 18, 19, 24, 29, 31, 44, 63, 82$ $\{1, 2, 3, 4, 6\}$-gesorteerd

De complexiteit van Shellsort (d.w.z. het aantal arrayvergelijkingen) hangt in hoge mate af van de gekozen stapgroottes. De analyse is in het algemeen extreem moeilijk en nog zeer incompleet.

1. Stapgroottes: $h_t = \lfloor \frac{n}{2} \rfloor$, $h_i = \lfloor \frac{h_{i+1}}{2} \rfloor$ voor $i = t-1, \ldots, 1$.
   Het aantal rondes is dan $t = \lfloor \lg n \rfloor$.
   Voor het gemak nemen we $n = 2^k$. Dan $t = \lg n = k$ en stapgroottes zijn: $2^{k-1}, 2^{k-2}, \ldots, 4, 2, 1$.

   Stelling A
   Het aantal arrayvergelijkingen dat Shellsort met deze serie stapgroottes doet is in de worst case $\Omega(n^2)$.

   Voor het bewijs (zie college) is het voldoende om een bad case aan te geven waarvoor het aantal vergelijkingen $\Omega(n^2)$ is.

Stelling B
Het aantal arrayvergelijkingen dat Shellsort doet met deze serie stapgroottes is in de worst case $O(n^2)$.

Bewijs: zie college 8.

Gevolg van A en B.
In de worst case doet Shellsort met Shell's stapgrootte-serie $\Theta(n^2)$ arrayvergelijkingen.

2. Stapgroottes (Hibbard): $2^k - 1, 2^{k-1} - 1, \ldots, 7, 3, 1$
   ($k = \lfloor \lg n \rfloor$).

   Stelling. In de worst case doet Shellsort met Hibbard's serie stapgroottes $O(n\sqrt{n})$ arrayvergelijkingen.

3. Het kan nog beter!

Stapgroottes (Hibbard): $2^k-1, 2^{k-1}-1, \ldots, 7, 3, 1$ ($k = \lfloor \lg n \rfloor$). Merk op: opeenvolgende stapgroottes zijn hier relatief priem (hebben geen gemeenschappelijke deler). Dit volgt uit: $h_{i+1} = 2h_i+1$.

Stelling. In de worst case doet Shellsort met Hibbard's reeks stapgroottes $O(n\sqrt{n})$ arrayvergelijkingen.

Bewijsschets: Je kunt inzien dat je bij stapgrootte h_i voor elk van de $n-h_i$ getallen maar met hoogstens $8h_i+4$ getallen hoeft te vergelijken*. Met $h_i = 3$, $h_{i+1} = 7$ en $h_{i+2} = 15$ weet je bij het h_i-sorteren bijvoorbeeld al zeker dat $A[40] \leq A[92]$ omdat $A[40] \leq A[47] \leq A[62] \leq A[77] \leq A[92]$. Dat is dus $O(nh_i)$ werk. We gebruiken dit voor $h_i < \sqrt{n}$. Voor $h_i > \sqrt{n}$ als voorheen (zie bewijs Stelling B): $O(n^2/h_i)$. Totaal geeft dat uiteindelijk $O(n\sqrt{n})$.

*dit volgt uit de speciale eigenschappen van de h_i

Bij bewijs Hibbard (college 8)*

We gebruiken een Stelling uit de getaltheorie

Als ggd$(h, \ell) = 1$, dan is het grootste gehele getal dat niet als $a \cdot h + b \cdot \ell$ ($a, b \geq 0$, geheel) geschreven kan worden: $m = h\ell - h - \ell$

Gevolg

Alle gehele getallen $\geq h\ell - h - \ell = (h-1)(\ell-1)$ kunnen als lineaire combinatie van h en ℓ worden geschreven.
Voorbeeld: $h = 3, \ell = 7$ dus $m = 12$: $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, \ldots$

Als we dit toepassen op h_{i+1} en h_{i+2} vinden we: alle integers j met $j \geq (h_{i+1}-1)(h_{i+2}-1) = 8{h_i}^2+4h_i$ kunnen geschreven worden als $a \cdot h_{i+1} + b \cdot h_{i+2}$
Voor deze j geldt dus bij het h_i-sorteren dat $A[p-j] = A[p-a \cdot h_{i+1} - b \cdot h_{i+2}] \leq A[p]$, want het array is op dat moment al h_{i+1}- en h_{i+2}-gesorteerd.
Voor elk van de $n - h_i$ in te voegen array-elementen hoeven we dus maar met maximaal $(8{h_i}^2 + 4h_i)/h_i = 8h_i + 4$ elementen te vergelijken.

*dit is extra voor de liefhebbers

Voorbeeld. Na 8-sorteren heeft 3-sorteren i.h.a. veel meer effect dan 4-sorteren.

8-gesorteerd, 36 inversies

$5, 2, 11, 7, 15, 3, 16, 6, 12, 9, 14, 8, 19, 13, 20, 10$

$\downarrow$ 4-sorteren

31 inversies

$5, 2, 11, 6, 12, 3, 14, 7, 15, 9, 16, 8, 19, 13, 20, 10$

8-gesorteerd, 36 inversies

$5, 2, 11, 7, 15, 3, 16, 6, 12, 9, 14, 8, 19, 13, 20, 10$

$\downarrow$ 3-sorteren

7 inversies

$5, 2, 3, 6, 7, 8, 9, 12, 11, 10, 13, 15, 14, 16, 20, 19$

Betere stapgrootteseries:

- Twee (!) doorgangen: $h_2 \approx 1,72 \cdot \sqrt[3]{n},\ h_1 = 1$
  Gemiddeld $O(n^{5/3})$

- Heel veel doorgangen: increments van de vorm $2^i \cdot 3^j$:
  $1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27, 32, 36, 48, 54, 72, 81, \ldots$
  Worst case $O(n(\lg n)^2)$

- Increments van de vorm $4^{j+1} + 3 \cdot 2^j + 1$:
  $1, 8, 23, 77, 281, 1073, 4193, 16577, \ldots$
  Worst case $O(n^{4/3})$

- $\cdots$ (optimale serie stapgroottes is [nog] onbepaald)

Opgave

Neem aan dat n een macht van 7 is, dus $n = 7^k$ met $k \geq 0$ geheel.
Bewijs nu dat het aantal vergelijkingen dat Shellsort met stapgroottes $n/7$, $n/49$, $n/343$, . . . , 49, 7, 1 in de worst case doet $\Omega(n^2)$ is.

Het bewijs gaat analoog aan het bewijs voor Shellsort met Shell's rijtje stapgroottes.

- **Volgende college:** vrijdag 4 april, 11.00 – 12.45, Gorlaeusgebouw zaal BW.0.40

- **Straks werkcollege:** vrijdag 21 maart, 13.15 – 15.00, Gorlaeusgebouw zalen BW.0.29 en BW.0.30
  Opgaven: 39 (geen Heapsort), 40, 41abc(d), 42, 43

- Vrijdag 4 april **geen werkcollege**

- **Tweede huiswerkopgave:**
  deze week op de website

- **Website:**
  liacs.leidenuniv.nl/~graafjmde/COMP/