Zevende college complexiteit

17 maart 2008

Ondergrens sorteren, Quicksort

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

Aannames:

- A bevat n verschillende waarden.
- het sorteeralgoritme stopt zodra de sortering (onderlinge ordening) gevonden is.

Zo'n algoritme correspondeert (voor elke n) met een beslissingsboom die de series vergelijkingen representeert die het algoritme uitvoert voor elke mogelijke invoer.

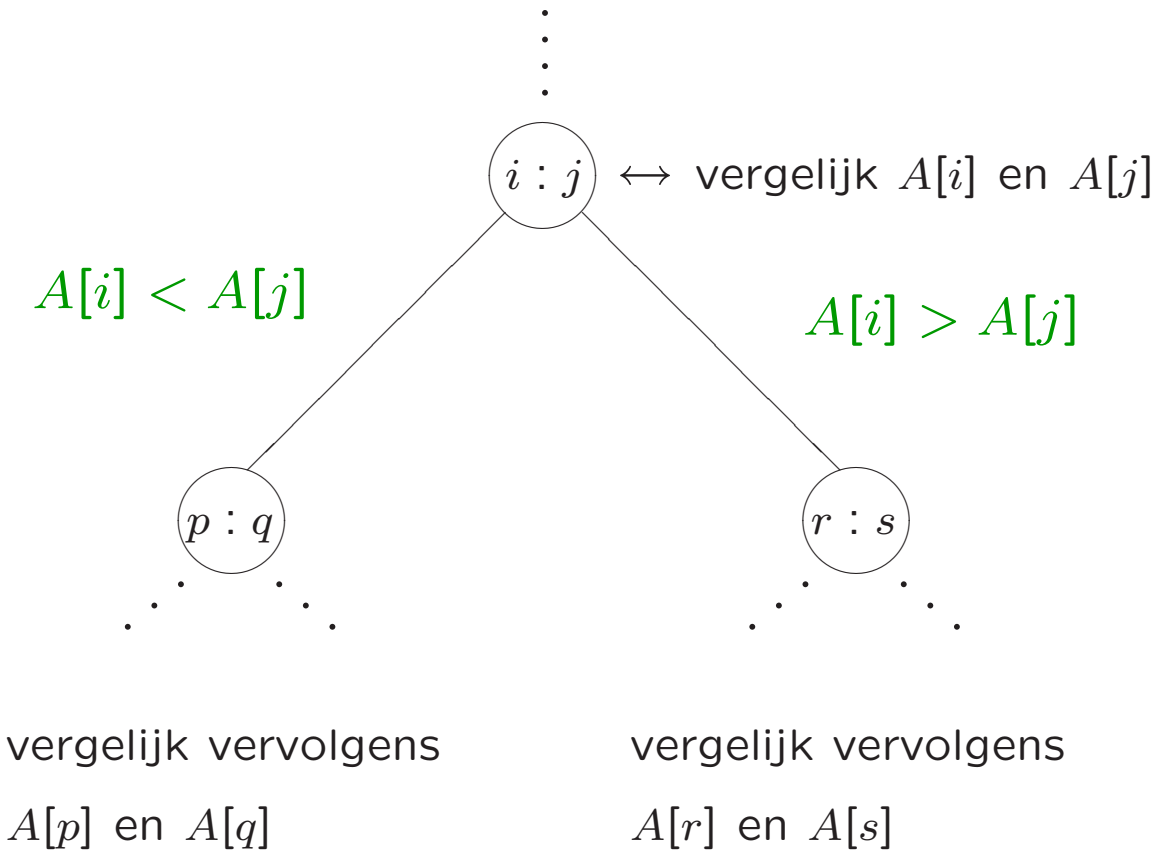
algoritme gebaseerd op het doen van arrayvergelijkingen $A[i] < A[j]$

$$\Downarrow$$

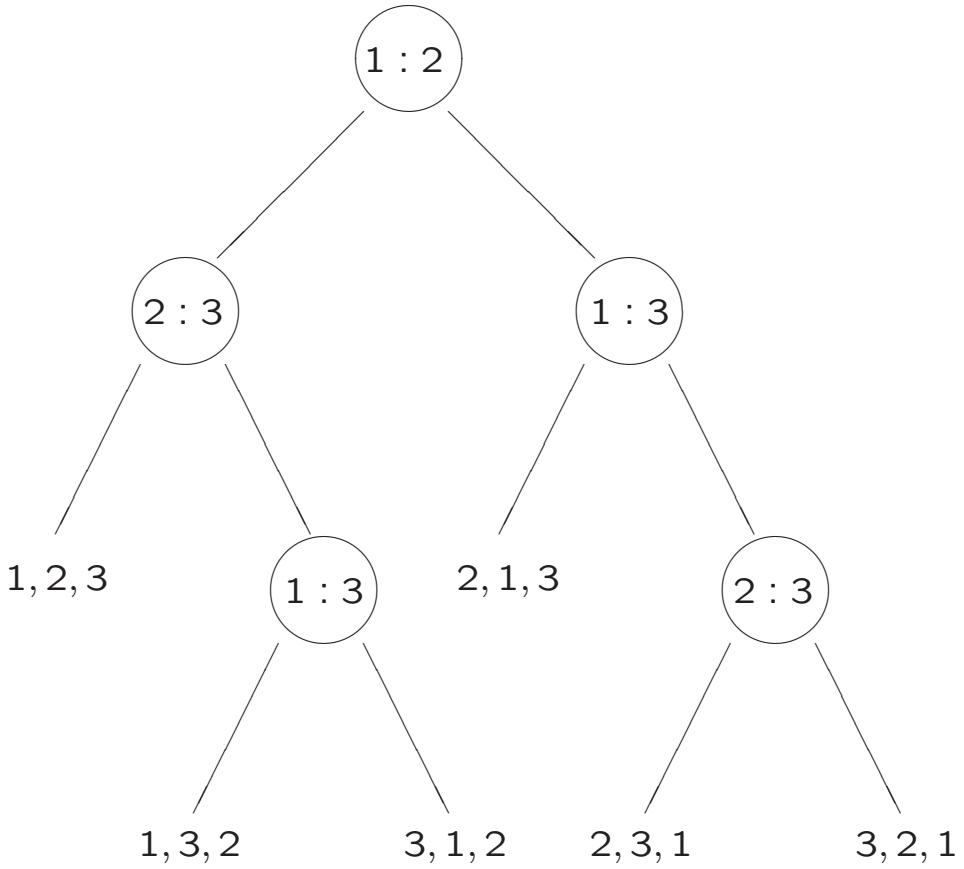
beslissingsboom*: binaire boom waarin de interne knopen corresponderen met arrayvergelijkingen en de bladeren/externe knopen met het eindresultaat†; een pad vanaf de wortel naar een blad correspondeert met een executie van het algoritme

*alle $A[i]$ zijn verschillend

†eindresultaat = de gevonden sorteringen/ordeningen (in dit geval)



Beslissingsboom voor algoritmen gebaseerd op arrayvergelijkingen



Beslissingsboom voor Insertion sort met $n = 3$

1. - alleen de onderlinge volgorde van de array-elementen wordt onderscheiden; niet de waarde
   - het rijtje $3, 15, 8, 11, 6$ wordt bijvoorbeeld precies zo behandeld door het sorteeralgoritme als het rijtje $1, 5, 3, 4, 2$
   - 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

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 heeft precies $n!$ bladeren (n = aantal array-elementen)

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

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.

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:

$$E = \Sigma_{\text{bladeren}} \text{lengte pad wortel naar blad}$$

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

$$E \geq b(\lg b - 1)$$

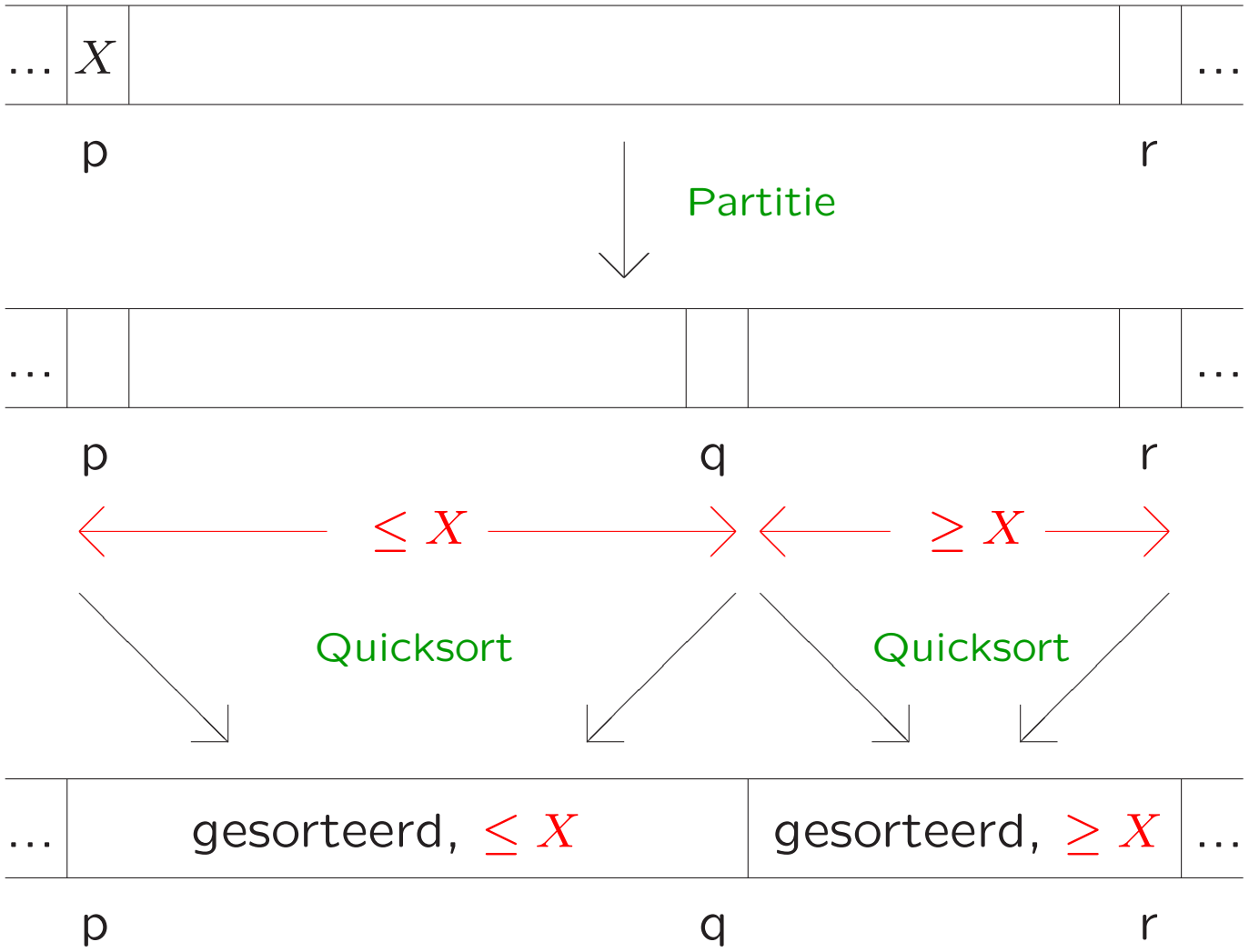
Gevolg. De gemiddelde lengte van een pad van de wortel naar een blad $= \frac{E}{b} \geq \lg b - 1$.

Quicksort (Hoare 1962)

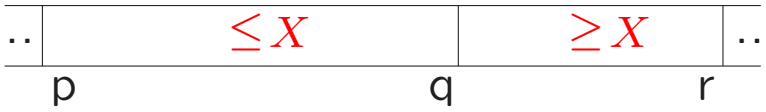
```
QuickSort(A, p, r)::
// sorteert A[p], ..., A[r] oplopend
        if p < r then
                q := Partitie(A, p, r);
                QuickSort(A, p, q);
                QuickSort(A, q + 1, r);
        fi
```

Aanroep:
Quicksort(A,1,n)

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


... X ...
p
r
Partitie
...
...
p
q
r
≤ X
≥ X
Quicksort
Quicksort
... gesorteerd, ≤ X gesorteerd, ≥ X ...
p
q
r

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

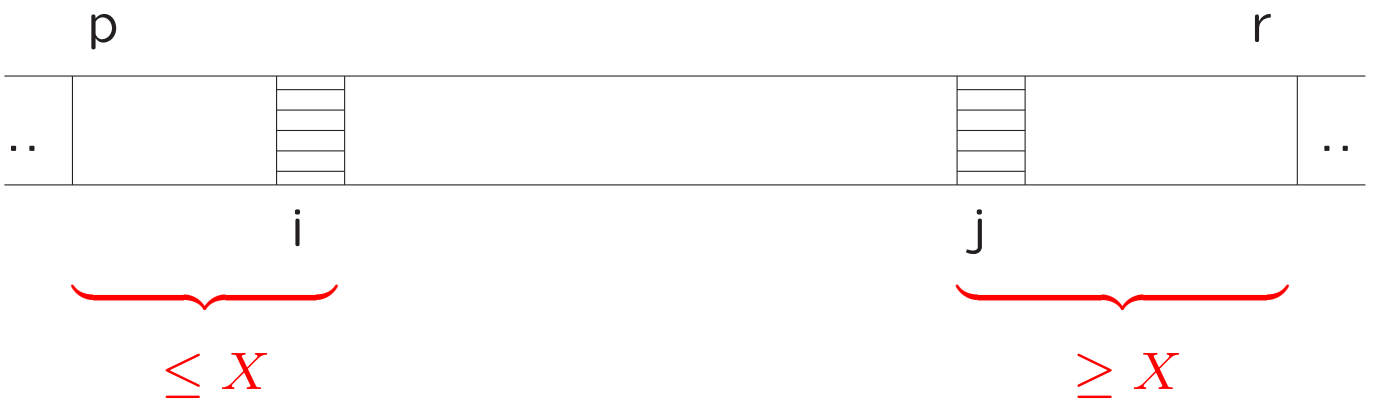


(*) // hier komt nog wat
$x = A[p]$; $i := p - 1$; $j := r + 1$;
while $i < j$ **do**
    $j := j - 1$; // loop met j naar links
    while $A[j] > x$ **do**
        $j := j - 1$;
    od // tot je een waarde $A[j] \leq x$ vindt
    i := i + 1; // loop met i naar rechts
    while $A[i] < x$ **do**
   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

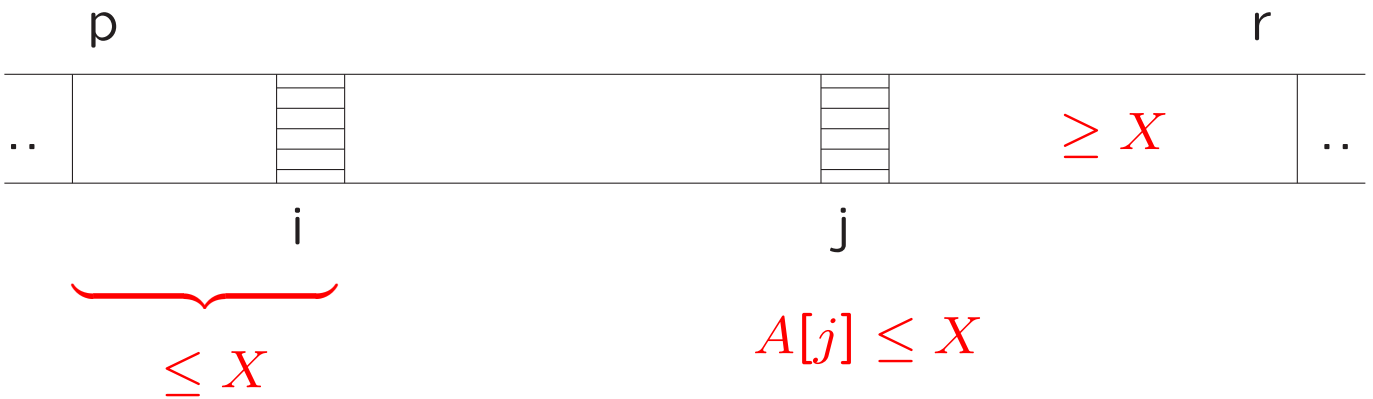
Opmerkingen bij Partitie

1. De *basisoperatie* is het vergelijken van array-elementen: $A[j] > x$ en $A[i] < x$
2. Partitie stopt met $i = j$ of $i = j + 1$
3. Na afloop is altijd $j \geq p$ en $j \leq r - 1$, dus $p \leq q \leq r - 1$. Quicksort wordt dus op echt kleinere rijtjes recursief aangeroepen
4. Elk array-element wordt precies één keer met x vergeleken, behalve $A[q]$ (twee keer) en eventueel $A[q + 1]$ (soms twee keer)
5. Partitie doet altijd $\Theta(m)$ vergelijkingen, nl. $m + 1$ of $m + 2$, met m het aantal elementen van het (deel)array $A[p], \ldots, A[r]$
6. Er worden elementen verwisseld die ver uit elkaar kunnen liggen. Per vergelijking worden dus wellicht > 1 inversies opgeheven

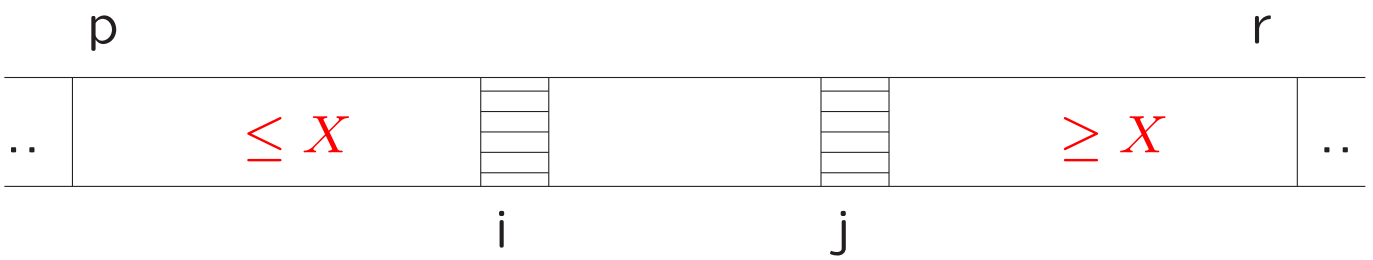
1. Na een volledige ronde:



2. Na de j-loop:

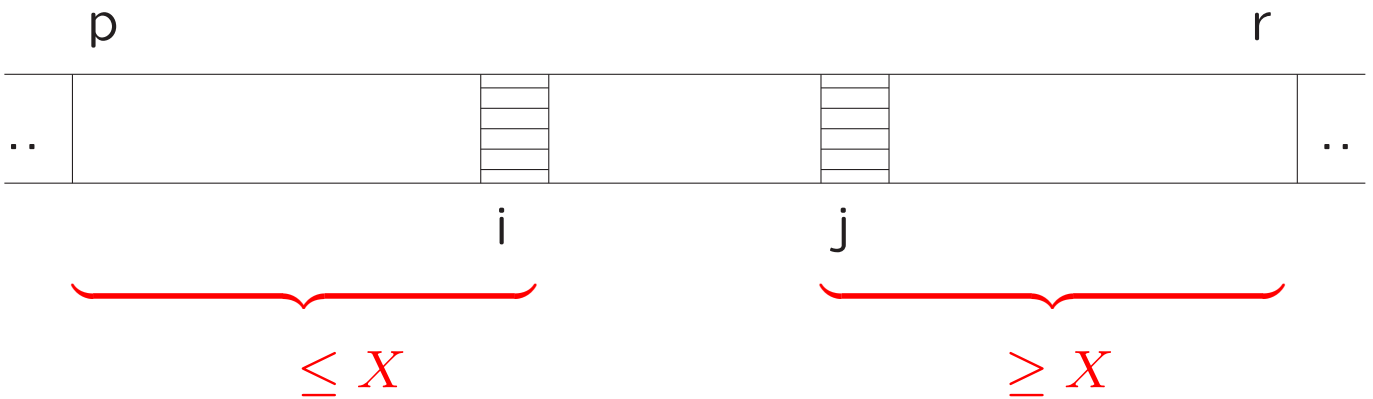


3. Na de i-loop, vóór de verwisseling:



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

4. Na de verwisseling:



1. Wat gebeurt er met gelijke array-elementen?

2. Wat gebeurt er met een rijtje met allemaal verschillende elementen?

3. Stel dat we $A[i]$ en $A[i+k]$, die verkeerd om staan, verwisselen. Hoeveel inversies worden dan ten minste resp. ten hoogste opgeheven?

1. Bad case voor Quicksort:

   Op het reeds gesorteerde rijtje (!), bijvoorbeeld $1, 2, \ldots, n$, doet Quicksort $\sum_{k=3}^{n+1} k = \frac{1}{2}n(n+3) - 2 = \Theta(n^2)$ vergelijkingen.

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

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

   Het aantal vergelijkingen $B(n)$ 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 vergelijkingen voorstellen dat Quicksort doet in de worst case. Dan voldoet W aan*:

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

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

Samen met de bad case hebben we dus:
Stelling.
Quicksort doet $\Theta(n^2)$ vergelijkingen in de worst case

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

De keuze van de pivot (spil; x dus) heeft grote invloed op de complexiteit van Quicksort. Standaard het eerste array-element ($A[p]$) als pivot kiezen 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 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 $O(n \lg n)$ vergelijkingen in de average case.

*Randomized 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

Algorithm	Worst case	Average	Space usage
Insertion sort	$n^2/2$	$\Theta(n^2)$	In place
Quicksort	$n^2/2$	$\Theta(n \lg n)$	Extra space proportional to $\lg n$
Mergesort	$n \lg n$	$\Theta(n \lg n)$	Extra space proportional to n for merging
Heapsort	$2n \lg n$	$\Theta(n \lg n)$	In place
Accel. Heapsort	$n \lg n$	$\Theta(n \lg n)$	In place

Vergelijking van verschillende sorteeralgoritmen

(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
10000	58.864	0.42998	0.71650	0.36765	0.31532
100000	NA	5.7298	8.8591	4.2298	3.5882
1000000	NA	71.164	104.68	47.065	41.282

Quicksort* = optimized Quicksort