Achtste college algoritmiek

2 april 2025

Verdeel en Heers

Puzzel 2:

$$\begin{array}{r} \mathrm{D\ O\ N\ A\ L\ D} \\ \mathrm{G\ E\ R\ A\ L\ D} \\ \hline \mathrm{R\ O\ B\ E\ R\ T} \end{array} +$$

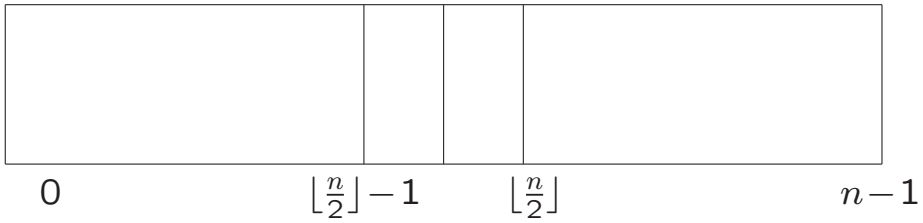
Elke letter stelt een cijfer voor $(0, 1, \ldots, 9)$ en verschillende letters corresponderen met verschillende cijfers. Het cijfer 0 komt niet voor als meest linkse cijfer in een getal. (Dus i.h.b. is $\mathrm{D} \neq 0$, $\mathrm{G} \neq 0$ en $\mathrm{R} \neq 0$.)

Om het makkelijker te maken is gegeven dat $\mathrm{D} = 5$.

Vergelijk ook Levitin, hoofdstuk 3.4, opgave 11.

Wordt programmeeropdracht 2, later deze week.

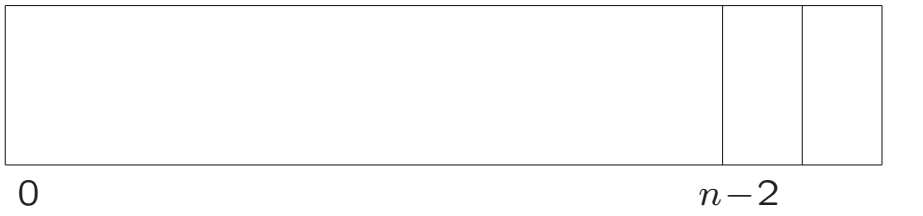
Divide and conquer



Mergesort

Quicksort

Decrease and conquer (decrease by one)



Insertion sort

De sorteermethode Quicksort is, evenals Merge sort, gebaseerd op het verdeel en heers principe. Quicksort wordt in de praktijk veel gebruikt, omdat het (gemiddeld) zeer efficiënt sorteert.

Quicksort($A[l \cdots r]$)::

```
Quicksort(A[l···r])::
// sorteert het (sub)array A[l···r] recursief
// uitvoer: A[l···r] oplopend gesorteerd
    if l < r
        s :=Partitie(A[l···r]); // s het splitspunt
        Quicksort(A[l···s − 1]);
        Quicksort(A[s + 1···r]);
    fi .
```

Voorbeeld: 5 3 1 9 8 2 4 7

Partitie

Partitie($A[l \cdots r]$) ::

// partitioneert een (sub)array, met $A[l]$ als spil (pivot)

$p := A[l]$;

$i := l; j := r + 1$;

repeat

repeat $i := i + 1$; **until** $i > r$ **or** $A[i] \geq p$;

repeat $j := j - 1$; **until** $A[j] \leq p$;

if $i < j$ **then**

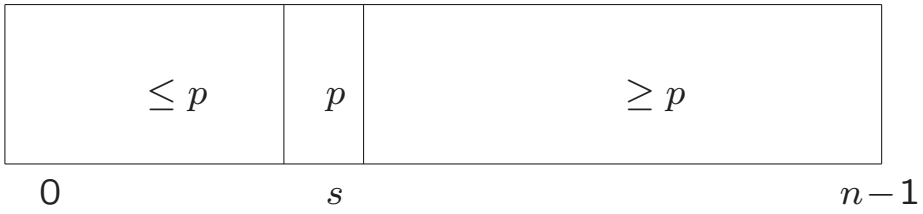
Wissel($A[i], A[j]$);

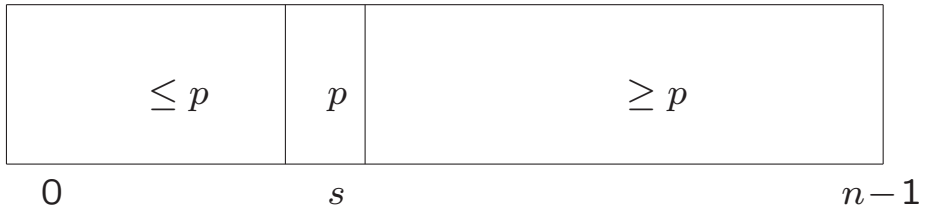
if

until $i \geq j$;

Wissel($A[l], A[j]$);

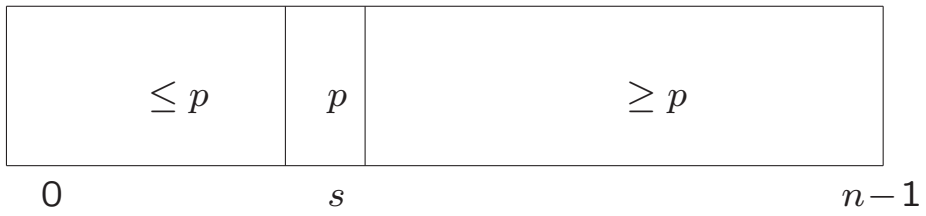
return j; .





Quicksort

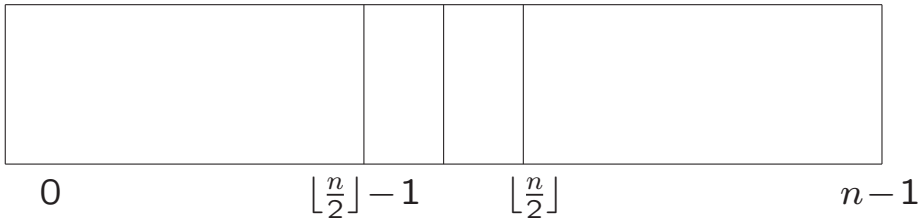
- ruimte complexiteit: . . .
- best case tijd: . . .
  wat zijn best cases?
- worst case tijd: . . .
  wat zijn worst cases?
- aantal worst cases. . .
- average case tijd: . . .



Quicksort

- ruimte complexiteit: $O(\log n)$
- best case tijd: $\Theta(n \log n)$
- worst case tijd: $\Theta(n^2)$
- aantal worst cases: 2^{n-1}
- average case tijd: $\Theta(n \log n)$

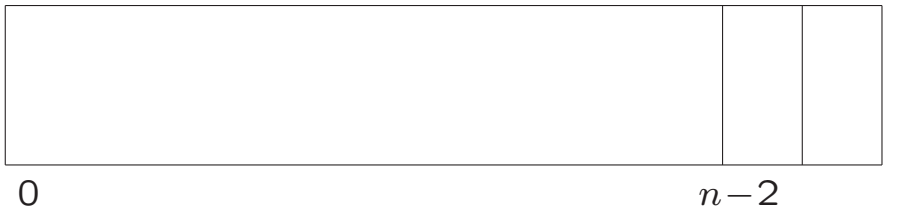
Divide and conquer



Mergesort

Quicksort

Decrease and conquer (decrease by one)



Insertion sort

DECREASE by one & CONQUER

Insertionsort($A[0 \cdots m-1]$)::
  if $m > 1$
    Insertionsort($A[0 \cdots m-2]$);
    Voeg $A[m-1]$ op de juiste plek in;
  fi .

Invoegen van $A[m-1]$ in het reeds gesorteerde voorstuk $A[0] \cdots A[m-2]$ door van rechts naar links $A[m-1]$ te vergelijken met $A[i]$. Deze recursieve versie komt overeen met de iteratieve versie zoals bij Programmeermethoden behandeld (zie ook Levitin):

$$A[0] \leq A[1] \leq \ldots \leq A[i] \leq A[i+1] \leq \cdots \leq A[m-3] \leq A[m-2] || A[m-1] \cdots$$

kleiner of gelijk $A[m-1]$ ↑ groter dan $A[m-1]$

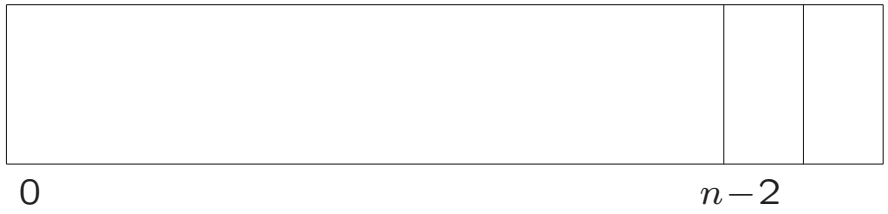
hier invoegen

Voorbeeld: 5 3 1 9 8 2 4 7

Iteratief

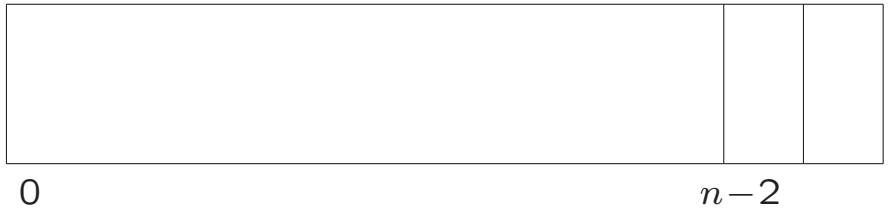
Insertionsort($A[0 \cdots n-1]$)::

```
    for i = 1 to n − 1 // A[0..i − 1] is al gesorteerd
        getal = A[i];
        j = i − 1;
        while j ≥ 0 and getal < A[j]
            A[j + 1] = A[j]; // schuif A[j] naar rechts
            j − −;
        od
        A[j + 1] = getal; // j is 1 'te laag' geworden
    od
```



Insertion sort

- ruimte complexiteit: . . .
- worst case tijd: . . .
  wat zijn worst cases?
- aantal worst cases: . . .
- average case tijd: . . .
- best case tijd: . . .
  wat zijn best cases?



Insertion sort

- ruimte complexiteit: iteratief $O(1)$
- worst case tijd: $\Theta(n^2)$
- aantal worst cases: 1 of 2^{n-1}
- average case tijd: $\Theta(n^2)$
- best case tijd: $\Theta(n)$

Mergesort:

- worst case complexiteit: $\Theta(n \log n)$
- extra geheugen: $O(n)$

Quicksort:

- worst case complexiteit: $\Theta(n^2)$ voor (o.a.) het reeds gesorteerde rijtje
- average case complexiteit: $\Theta(n \log n)$
- extra geheugen: $O(\log n)$

Insertion sort:

- worst case/average case complexiteit: $\Theta(n^2)$
- extra geheugen: in situ

Vermenigvuldiging van grote integers:

Het voor de hand liggende algoritme gebruikt voor de vermenigvuldiging van twee getallen bestaande uit n-cijfers (digits) n^2 digit-vermenigvuldigingen

Voorbeeld (n=2): $12 * 34 = \ldots$

Vermenigvuldiging van grote integers:

Het voor de hand liggende algoritme gebruikt voor de vermenigvuldiging van twee getallen bestaande uit n-cijfers (digits) n^2 digit-vermenigvuldigingen Het kan echter op magische wijze beter (althans voor zeer grote getallen) via divide and conquer. Gebruik een generalisatie van de volgende truc (met $n = 2$):

$$c = a * b = (a_1 10^1 + a_0) * (b_1 10^1 + b_0) = c_2 10^2 + c_1 10^1 + c_0$$

$$c_2 = a_1 * b_1$$

$$c_0 = a_0 * b_0$$

$$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$$

Voor $n = 2$ zijn hier dus 3 i.p.v. 4 digit-vermenigvuldigingen gebruikt!

Voorbeeld $n = 8$:

$$87593264 * 49367251 =$$

$$(8759 \cdot 10^4 + 3264) * (4936 \cdot 10^4 + 7251) = c_2 10^8 + c_1 10^4 + c_0$$

$$c_2 = 8759 * 4936$$

$$c_0 = 3264 * 7251$$

$$c_1 = 8759 * 7251 + 3264 * 4936 =$$

$$(8759 + 3264) * (4936 + 7251) - (c_2 + c_0)$$

Voorbeeld $n = 8$:

$$87593264 * 49367251 =$$
$$(8759 \cdot 10^4 + 3264) * (4936 \cdot 10^4 + 7251) = c_2 10^8 + c_1 10^4 + c_0$$

$$c_2 = 8759 * 4936$$
$$c_0 = 3264 * 7251$$
$$c_1 = 8759 * 7251 + 3264 * 4936 =$$
$$(8759 + 3264) * (4936 + 7251) - (c_2 + c_0)$$

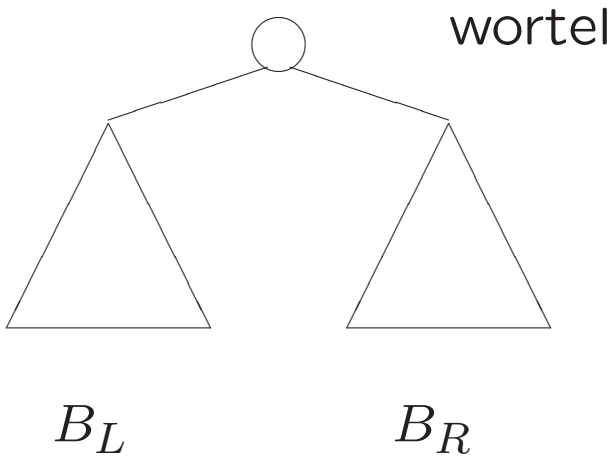
Het vermenigvuldigen van twee getallen bestaande uit $n = 2^k$ bits is zo teruggebracht tot 3 keer hetzelfde probleem voor $n/2 = 2^{k-1}$. Als $M(n)$ het aantal digitvermenigvuldigingen is voor $n = 2^k$, dan voldoet $M(n)$ aan:

$$M(n) = 3 * M(n/2) \text{ als } n > 1; M(1) = 1,$$

en vinden we: $M(n) = n^{\lg 3} < n^{\lg 4} = n^2$.

Een binaire boom B wordt recursief gedefinieerd als ofwel leeg, ofwel bestaande uit een knoop (de wortel) en twee disjuncte subbomen B_L en B_R die beide ook weer een binaire boom zijn: de linkersubboom en de rechtersubboom.

$B =$



Bij (veel) problemen met binaire bomen ligt oplossen via divide & conquer dus voor de hand.

De hoogte van een binaire boom is het hoogste niveau dat voorkomt, waarbij de wortel per definitie op niveau 0 zit.

Voor de hoogte van een binaire boom B geldt dus:

$$\text{hoogte}(B) = 1 + \texttt{max}\ \{\ \text{hoogte}(B_L),\ \text{hoogte}(B_R)\ \}$$

Verdeel en heers algoritme in C++:

```
int hoogte( knoop * root ) {
  if ( root == nullptr )      // lege boom
    return -1;
  else
    return ( 1 + max( hoogte( root->links ), hoogte( root->rechts ) ) );
}  // hoogte
```

DECREASE by one & CONQUER

Insertionsort($A[0 \cdots m-1]$)::
  if $m > 1$
    Insertionsort($A[0 \cdots m-2]$);
    Voeg $A[m-1]$ op de juiste plek in;
  fi .

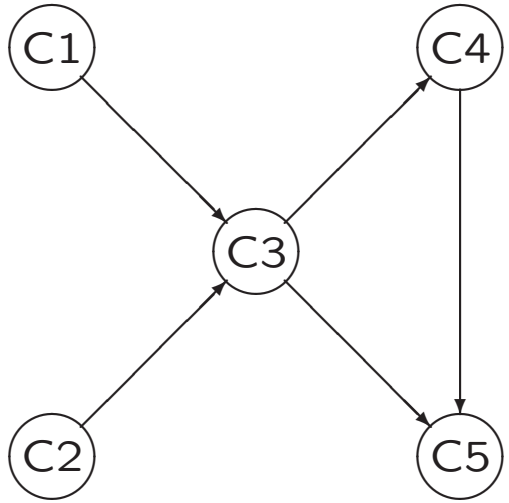
Invoegen van $A[m-1]$ in het reeds gesorteerde voorstuk $A[0] \cdots A[m-2]$ door van rechts naar links $A[m-1]$ te vergelijken met $A[i]$. Deze recursieve versie komt overeen met de iteratieve versie zoals bij Programmeermethoden behandeld (zie ook Levitin):

$$A[0] \leq A[1] \leq \ldots \leq A[i] \leq A[i+1] \leq \cdots \leq A[m-3] \leq A[m-2] || A[m-1] \cdots$$

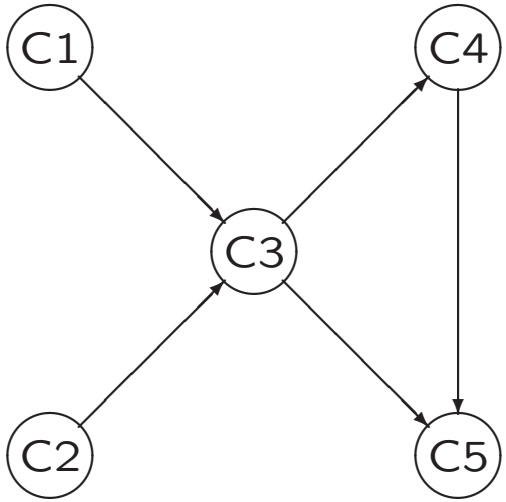
kleiner of gelijk $A[m-1]$ ↑ groter dan $A[m-1]$

hier invoegen

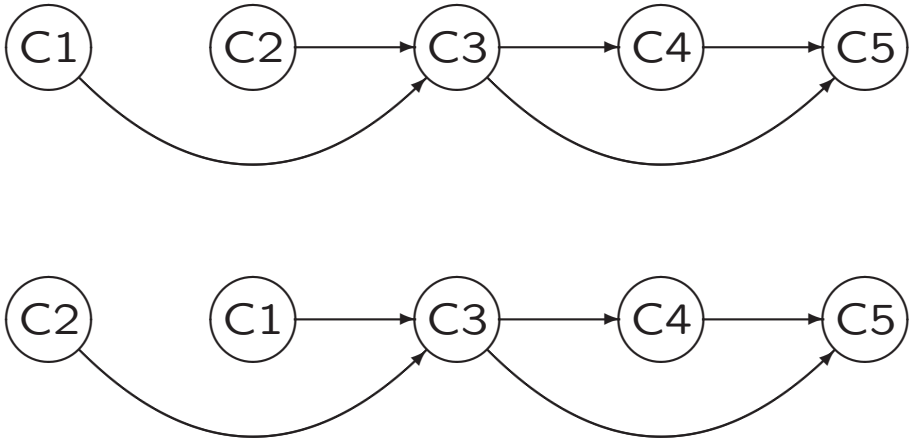
Gerichte grafen, b.v. met benodigde voorkennis voor cursussen:



Gerichte grafen, b.v. met benodigde voorkennis voor cursussen:



Mogelijke volgordes



Gerichte grafen

- eis aan graaf
- algoritme
- implementatie
- efficiëntie
- ander algoritme
- efficiëntie
- effect algoritmes als graaf cyclisch is

Gerichte grafen

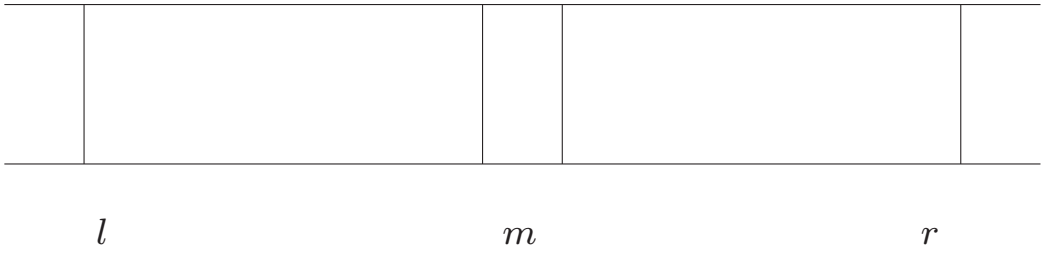
- eis aan graaf: acyclisch
- algoritme: source-removal
- implementatie: aantal inkomende takken per knoop bijhouden
- efficiëntie: $\Theta(n+m)$
- ander algoritme: DFS
- efficiëntie: $\Theta(n+m)$
- effect algoritmes als graaf cyclisch is: verschillend

Zie paragraaf 4.2 van boek + antwoorden werkcollege 8.

Binair zoeken is een voorbeeld van decrease and conquer, decrease by a constant factor. Hier de iteratieve versie.

// invoer: oplopend gesorteerd array $A[0..n-1]$, te zoeken waarde K
// uitvoer: positie van K in A (-1 als K niet in A zit)

$l := 0; r := n-1;$
while $l \leq r$ **do**
  $m := \lfloor \frac{l+r}{2} \rfloor;$
  if $K = A[m]$ **then** // gevonden
    return m;
  else if $K < A[m]$ **then** // links verder zoeken
    $r = m - 1;$
  else // rechts verder zoeken
    $l = m + 1;$ **fi**
  fi
od
return -1;



altijd *links óf rechts* verdergaan

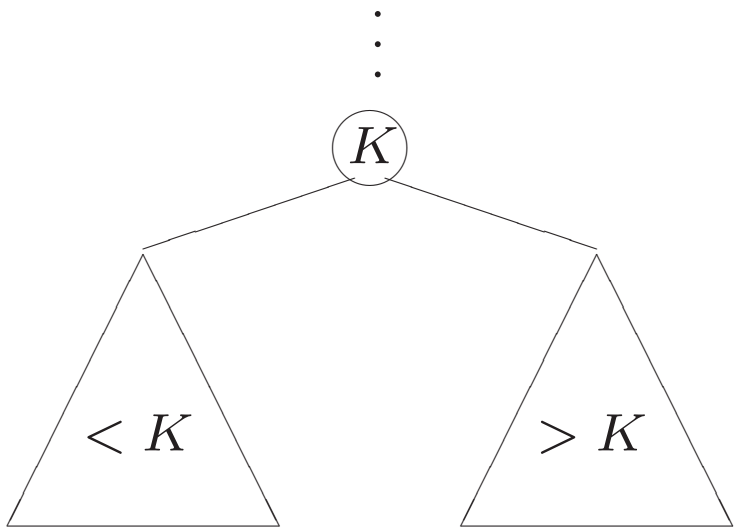
Variable-size decrease: de reductie in grootte is variabel,
dus kan in elke stap anders zijn

Voorbeelden:

- Algoritme van Euclides
  Dit is gebaseerd op: ggd(m, n) = ggd(n, m **mod** n)
- Flipping pancakes (Levitin, excercise 4.5.12)

- Binaire zoekbomen

Een binaire zoekboom is een binaire boom waarbij voor elke knoop geldt dat de waarde in die knoop groter is dan alle waarden in zijn linkersubboom, en kleiner dan alle waarden in zijn rechtersubboom.



Bij het zoeken naar een waarde in een gewone binaire boom (bijv. WLR) moeten in het slechtste geval alle n knopen bekeken worden.
Zoeken in een binaire zoekboom is i.h.a. efficiënter: in het slechtste geval worden $h+1$ knopen bekeken, met h de hoogte van de boom.

```
knoop* zoeken(knoop* root, int getal) {
  if ( root == nullptr )      // lege boom
    return nullptr;
  else
    if ( root->info == getal )       // gevonden!
      return root;
    else
      if ( getal < root->info )
        return zoeken(root->links, getal);
      else
        return zoeken(root->rechts, getal);
}  // zoeken
```

- **Lezen/leren bij dit college:**
  5.2, 5.3, 4.inl, 4.1, 4.2, 4.4 (geen Russian peasant), 4.5 (inl. + Binary Search Tree)

- **Werkcollege** verdeel en heers
  vrijdag 4 april, 11.00–12.45, BW.0.07, BW.0.08

- **Opgaven:**
  zie `http://www.liacs.leidenuniv.nl/~vlietrvan1/algoritmiek/`

- **Practicumbijeenkomst programmeeropdracht 1/2:**
  dinsdag 8 april 2025, 09.00-10.45, computerzalen DM.0.09, DM.0.13

- **Volgend college:**
  woensdag 9 april 2025, 15.15–17.00