Complexiteit

College 4

Docent: Lieuwe Vinkhuijzen

17 februari 2020

Vorige week

- 2-CNF-SAT opgelost met Bereikbaarheid
- Beslissingsbooom argument voor zoeken in $\Omega(\log(n))$
- Adversary argument voor sorteren in $\Omega(n \log(n))$

Deze week

1. Reachability oplossen met 2-CNF-SAT
2. Zoeken in een array
3. Sorteren
4. Beslissingsboom argument voor sorteren in $\Omega(n \log(n))$

Gerichte bereikbaarheid

Input: Een gerichte graaf $G = (V, E)$, twee knopen $s, t \in V$
Output: Is er een pad $s \rightarrowtail t$ in G?

2-CNF Satisfiability

Input: Een Boolse formule, ϕ, in 2-CNF
Output: Bestaat er een toekenning aan de variabelen z.d.d.[1] $\phi = \textsc{True}$?

Jargon: Als een toekenning x bestaat met $\phi(x) = \textsc{True}$ dan heet x een "satisfying assignment" en heet ϕ "satisfiable". Anders heet ϕ "unsatisfiable".

[1] z.d.d.: "Zodanig dat"

Bereikbaarheid oplossen met 2-CNF-SAT

Input: Een gerichte graaf $G = (V, E)$, twee knopen $s, t \in V$
Output: Is er een pad $s \rightarrowtail t$ in G?

Plan. We maken een 2-CNF formule ϕ die *satisfiable* is dan en slechts dan als t *niet* te bereiken is vanuit t.

1. Maak voor elke knoop $v \in V$ een variabele x_v
2. Maak voor elke tak $(a, b) \in E$ een clausule $(\neg x_a \vee x_b)$
3. Maak de clausules (x_s) en $(\neg x_t)$

Bereikbaarheid oplossen met 2-CNF-SAT

Input: Een gerichte graaf $G = (V, E)$, twee knopen $s, t \in V$
Output: Is er een pad $s \rightarrowtail t$ in G?

Zij ϕ de formule van de vorige frame.

Lemma

Voor elke goede toekenning x geldt $x_a \implies x_b$ dan en slechts dan als er een pad $a \rightarrowtail b$ is in G.

Proof.

($\Longleftarrow$) Als er in G een pad $s \to a \to b \to t$ is, dan geldt

$$x_s \implies x_a \implies x_b \implies \cdots \implies x_t$$

Dus er geldt ook

$$x_s \implies x_t$$

Bereikbaarheid oplossen met 2-CNF-SAT

Input: Een gerichte graaf $G = (V, E)$, twee knopen $s, t \in V$
Output: Is er een pad $s \rightarrowtail t$ in G?

Zij ϕ de formule van de vorige slide.

Lemma

Voor elke goede toekenning x geldt $x_a \implies x_b$ dan en slechts dan als er een pad $a \rightarrowtail b$ is in G.

Proof.

($\implies$) Als er géén pad $s \rightarrowtail t$ is in G, dan is er een satisfying assignment met $x_s = 1$ en $x_t = 0$. Die vind je zo:

1: $x_s := 1$
2: Propageer implicaties
3: Zet alle andere variabelen $x_v := 0$

Selectie in een 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)

[2] Blum, Floyd, Pratt, Rivest & Tarjan, 1973

Selectie in een 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)

Complexiteit
Het selectieprobleem is $O(n)$.
We bewijzen dit door een algoritme te geven voor dit probleem dat $O(n)$ vergelijkingen doet in de worst case[2].

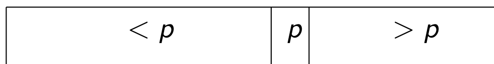
Merk op: Sorteren ligt voor de hand, maar kost $\Omega(n \log(n))$ tijd.

[2] Blum, Floyd, Pratt, Rivest & Tarjan, 1973

Selectie is $O(n)$

Plan: Verdeel en heers.

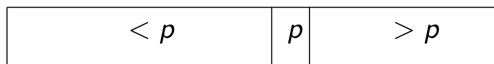
1: **procedure** $\textsc{Select}(k, A[1], \ldots, A[n])$
2: Kies een element $p := A[i]$, de *pivot* (bijvoorbeeld random)
3: Reorganiseer het array als volgt:



4: $j := index(p)$
5: **if** $j = k$ **then**
6: Gevonden!
7: **else if** $j < k$ **then**
8: $\textsc{Select}(k, A[1], \ldots, A[j])$ ▷ Recursie links
9: **else**
10: $\textsc{Select}(k - j, A[j + 1], \ldots, A[n])$ ▷ Recursie rechts
11: **end if**
12: **end procedure**

Selectie is $O(n)$

Plan: Verdeel en heers.



Vraag: Hoe goed is de pivot?

- **Best case:** We kiezen steeds per ongeluk de mediaan als pivot. Dan doen we $n + \frac{1}{2}n + \frac{1}{4}n + \frac{1}{8}n + \cdots \approx 2n$ vergelijkingen.
- **Worst case:** We kiezen steeds per ongeluk het grootste element, en doen $n + (n-1) + (n-2) + (n-3) + \cdots + (k) \approx \frac{1}{2}(n^2 - k^2)$ vergelijkingen, worst case $k = 1$.

Oplossing: Kies de mediaan als pivot.

Input: Een array getallen $A[1], \ldots, A[n]$
Output: De mediaan, d.w.z. de $\lceil \frac{n}{2} \rceil$-de grootste.

Mediaan is $O(n)$

1: **procedure** $\textsc{Mediaan}(A[1], \ldots, A[n])$
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 medianen[3]
5: Herorganiseer het array A rond m:

6: $i_m :=$ nieuwe index van m, i.e. $A[i_m] = m$
7: **if** $i_m \leq \frac{1}{2}m$ **then**
8: **return** $\textsc{Select}(\frac{n}{2}, A[1], \ldots, i_m)$
9: **else**
10: **return** $\textsc{Select}(\frac{n}{2} - i_m, A[i_m], \ldots, A[n])$
11: **end if**
12: **end procedure**

[3] m is niet per sé de mediaan van A

Selectie is $O(n)$

Vraag: Hoe goed is deze pivot m, de mediaan van de medianen?

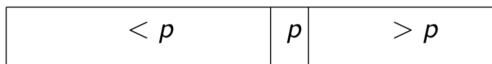
12	15	11	2	9	5	0	7	3	21	44	40	1	18	20	32	19	35	37	39
13	16	14	8	10	16	6	33	4	27	49	46	52	25	51	34	43	56	72	79
17	23	24	28	29	30	31	36	42	47	50	55	58	60	63	65	66	67	81	83
22	45	38	53	61	41	62	82	54	48	59	57	71	78	64	80	70	76	85	87
96	95	94	86	89	69	68	97	73	92	74	88	99	84	75	90	77	93	98	91

Blauw is de mediaan der medianen; groen is gegarandeerd lager, evenals het linker oranje gedeelte; rood en het rechter oranje stuk zijn gegarandeerd hoger. In het algemeen zijn gegarandeerd ongeveer $\frac{3n}{10}$ elementen uit het array kleiner dan de mediaan der medianen en gegarandeerd ongeveer $\frac{3n}{10}$ elementen groter. In het ergste geval ga je de recursie dus in op de resterende 70% van het array.

Selectie is $O(n)$

Plan: Verdeel en heers.

1: **procedure** $\textsc{Select}(k, A[1], \ldots, A[n])$
2: Kies een element $p := \textsc{Mediaan}(A[1], \ldots, A[n])$
3: Reorganiseer het array als volgt:



4: $j := index(p)$
5: **if** $j = k$ **then**
6: Gevonden!
7: **else if** $j < k$ **then**
8: $\textsc{Select}(k, A[1], \ldots, A[j])$ ▷ Recursie links
9: **else**
10: $\textsc{Select}(k - j, A[j + 1], \ldots, A[n])$ ▷ Recursie rechts
11: **end if**
12: **end procedure**

Selectie is $O(n)$

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

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

Bewering:

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

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

Selectie is $O(n)$

Opgave 26:
Elk algoritme voor het selectieprobleem dat is gebaseerd op arrayvergelijkingen doet in de worst case ten minste $\lceil \frac{n}{2} \rceil$ van zulke vergelijkingen.
Gevolg:
Het selectieprobleem is $\Omega(n)$.

Sorteren

Input: Een array getallen $A[1], \ldots, A[n]$
Output: Een oplopend gesorteerde versie van A

Sorteren met Selection Sort

Input: Een array getallen $A[1], \ldots, A[n]$
Output: Een oplopend gesorteerde versie van A

1: **procedure** $\textsc{SelectionSort}(A[1], \ldots, A[n])$
2:     **for** $i = 1 \ldots n-1$ **do**
3:         $j := \min(A[i], \ldots, A[n])$
4:         Verwissel$(A[i], A[j])$
5:     **end for**
6: **end procedure**

Sorteren met Selection Sort

Input: Een array getallen $A[1], \ldots, A[n]$
Output: Een oplopend gesorteerde versie van A

1: **procedure** SELECTIONSORT($A[1], \ldots, A[n]$)
2: **for** $i = 1 \ldots n - 1$ **do**
3: $j := \min(A[i], \ldots, A[n])$
4: Verwissel($A[i], A[j]$)
5: **end for**
6: **end procedure**

Basisoperatie: Minimum vinden in $A[i], \ldots, A[n]$ kost $n - i$ operaties.

$$T(n) = \sum_{i=1}^{n-1} n - i = \frac{1}{2} n(n-1) \in \Theta(n^2)$$

Sorteren met MergeSort

Mergesort en Quicksort zijn sorteermethoden die allebei gebaseerd zijn 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 Combineer-stap, Quicksort in de Verdeel-stap. Beide sorteermethoden zijn gebaseerd op arrayvergelijkingen, maar doen geen buurverwisselingen, zoals Insertion sort en Bubblesort.

Sorteren met MergeSort

Het (recursieve) Mergesort algoritme:

MergeSort(A, p, r)::
// sorteert $A[p], \ldots, A[r]$

```
if p < r then
    q := ⌊(p+r)/2⌋;
    MergeSort(A, p, q);
    MergeSort(A, q+1, r);
    Merge(A, p, q, r);
fi
```

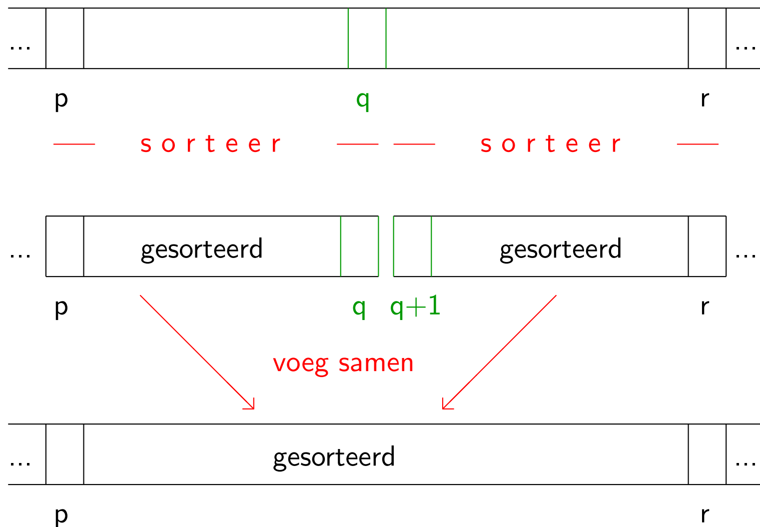
Aanroep: MergeSort$(A, 1, n)$.

verdeel

en

heers (voeg samen)

Sorteren met MergeSort



Sorteren met MergeSort

Merge(A, p, q, r)::

```
i := p; j := q + 1; k := p;
while i ≤ q and j ≤ r do
    if A[i] < A[j] then
        hulp[k] := A[i]; i := i + 1; k := k + 1;
    else
        hulp[k] := A[j]; j := j + 1; k := k + 1;
    fi
od
if i > q then  // eerste helft is op
    kopieer A[j], ..., A[r] naar hulp;
else           // tweede helft is op
    kopieer A[i], ..., A[q] naar hulp;
fi
kopieer hulp[p], ..., hulp[r] terug naar A;
```

Sorteren met MergeSort

- Merge(A,p,q,r) voegt de reeds gesorteerde deelrijtjes $A[p], \ldots, A[q]$ en $A[q+1], \ldots, A[r]$ samen tot een gesorteerd stuk $A[p], \ldots, A[r]$
- `hulp` is een hulparray ter grootte n (net als A)
- Geheel analoog kan een functie Merge(A, B, C, k, m) geschreven worden die twee gesorteerde rijen A (k elementen) en B (m elementen) samenvoegt tot de gesorteerde rij C ($n = k + m$ elementen)

Sorteren met MergeSort

- Voor het bepalen van de complexiteit van Merge tellen we het aantal vergelijkingen van de vorm: $A[i] < A[j]$
- Er worden altijd $2n$ verplaatsingen van array-elementen gedaan
- Is het aantal arrayvergelijkingen hier wel een goede maat voor de complexiteit?

Sorteren met MergeSort

Stel dat we met behulp van Merge twee gesorteerde rijtjes van respectievelijk k en m elementen (met $k + m = n$) samenvoegen tot één gesorteerde rij. Dan geldt:

1. Het aantal vergelijkingen in de worst case is $n - 1$
2. Het aantal vergelijkingen in de best case is $\min\{k, m\}$

Let op: *binnen Mergesort* is het aantal vergelijkingen een goede maat voor de complexiteit. Immers het aantal vergelijkingen is in dat geval altijd $\Theta(n)$, evenals het aantal verplaatsingen van array-elementen. In het algemene geval is dit niet zo (bijvoorbeeld $k = 1$ en $m = n - 1$).

Sorteren met MergeSort

Zij $T(n)$ = aantal vergelijkingen in de worst case van Mergesort op n elementen, met $n = 2^k$.

Dan geldt:

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T(\frac{n}{2}) + n - 1 & n = 2^k > 1 \end{cases}$$

Oplossing: $T(n) = n \lg n - n + 1 \in \Theta(n \lg n)$

Sorteren met MergeSort

Als n geen tweemacht is, wordt de recurrente betrekking:

$$T(n) = \begin{cases} 0 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n - 1 & n > 1 \end{cases}$$

Dan geldt eveneens: $T(n) \in \Theta(n \lg n)$.

Je kunt zelfs bewijzen: $T(n) = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1$

Mergesort is in orde van grootte optimaal voor wat betreft de worst case (immers: de ondergrens voor sorteren via arrayvergelijkingen is $\Omega(n \log n)$). Er is echter extra geheugenruimte ter grootte $\Theta(n)$ nodig.

Sorteren met MergeSort

Zij $B(n)$ = aantal vergelijkingen in de best case, met $n = 2^k$. Dan geldt:

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

Oplossing: $B(n) = \frac{n}{2} \log n \in \Theta(n \lg n)$.

Is MERGE optimaal?

Stelling

1. Elk algoritme gebaseerd op arrayvergelijkingen dat twee gesorteerde arrays (rijen) van lengte m samenvoegt tot één gesorteerd array, doet in het slechtste geval ten minste $2m - 1$ van zulke vergelijkingen.
   Voor $m = \frac{n}{2}$ (n even) is dit dus ten minste $n - 1$.
2. Voor het samenvoegen van twee rijtjes ter lengte $m - 1$ respectievelijk m is dat ten minste $2m - 2$.
   Voor $m = \lceil \frac{n}{2} \rceil$ (n oneven) is dit ten minste $n - 1$.

Gevolg. Binnen de klasse van samenvoegalgoritmen gebaseerd op arrayvergelijkingen is het beschreven Merge-algoritme optimaal, althans voor twee ongeveer even lange rijtjes.

Is MERGE optimaal?

We geven een klasse van invoerrijtjes waarop *elk* samenvoegalgoritme (gebaseerd op arrayvergelijkingen) *ten minste* $2m - 1$ vergelijkingen moet doen. Dat bewijst dan de stelling.
Kies stijgende rijtjes $A = (a_1, a_2, \ldots, a_m)$ en $B = (b_1, b_2, \ldots, b_m)$ zó dat alle a_i en b_j verschillend zijn en $a_i < b_j \leftrightarrow i < j$:

$$b_1 < a_1 < b_2 < \cdots < a_{i-1} < b_i < a_i < b_{i+1} < \cdots < b_m < a_m$$

Dan *moet* elk samenvoegalgoritme a_i met b_i vergelijken $(i = 1, 2, \ldots, m)$ en a_i met b_{i+1} $(i = 1, 2, \ldots, m - 1)$.
Het bewijs van deze bewering gaat uit het ongerijmde.

Bewijs

Elk werkend algoritme voor het selectieprobleem moet voor van tevoren onbekende rijtjes altijd *alle* n array-elementen bekeken (= vergeleken met een ander array-element) hebben.
Immers, veronderstel dat dat niet zo is, dus stel dat er een algoritme is dat voor zekere invoer A minstens één array-element niet bekijkt. Dan leidt dit als volgt tot een tegenspraak.
Stel dat je algoritme $A[j]$ als k-de in grootte aanwijst, maar $A[l]$ nooit vergeleken heeft. Dan kunnen we twee rijtjes B en C construeren, waarbij het algoritme op minstens één van de twee een verkeerd antwoord geeft. We nemen als invoerrijtjes B en C rijtjes die gelijk zijn aan A, met als enig verschil dat $B[l]$ groter is dan alle elementen van A en $C[l]$ kleiner dan alle elementen van A.
Het algoritme doet noodzakelijkerwijs precies hetzelfde op B en C als op A, want de vergeleken waardes zijn precies hetzelfde. Het geeft dus ook voor *beide* rijtjes het j-de element als de k-de in grootte: dus $B[j]$ wordt aangewezen als de k-de in grootte van B en $C[j]$ als de k-de in grootte van C. Dit kan echter nooit: als $B[j]$ de k-de in grootte uit B is, dan zijn er behalve $B[l]$ slechts $k-2$ elementen groter, dus er zijn in totaal ook maar $k-2$ elementen groter dan $C[j]$, en derhalve kan $C[j]$ niet de k-de in grootte van C zijn. Idem omgekeerd. Deze $A[j]$, het k-de grootste element in A (zo aangewezen door het algoritme), kan niet de k-de grootste in *zowel* B als C zijn. Dus tegenspraak met de correctheid van het algoritme. De aanname dat minstens één element niet bekeken is kan derhalve niet waar.

Ten slotte: om *alle* elementen te bekijken zijn minstens $\lceil \frac{n}{2} \rceil$ arrayvergelijkingen nodig.

Ondergrens voor Sorteren

(Op het bord)

Twee andere voorbeelden van sorteeralgoritmen met average case complexiteit $\Theta(n^2)$ zijn Selection sort en Bubble sort
Selection sort is een van de meest trivale algoritmen:

```
(1)   for i := 1 to n do
        // zoek het minimum van A[i ... n]
(2)       j = Minimum(A[i ... n]);
(3)       Verwissel(A[i], A[j]);
(4)   od
```

Selection sort vereist altijd
$(n-1)+(n-2)+\ldots+2+1=\frac{n}{2}(n-1)=\Theta(n^2)$
arrayvergelijkingen

Bubble sort gaat herhaaldelijk de lijst door en verwisselt elementen die onderling "verkeerd" staan.

```
(1)   boven := n; gewisseld := true;
(2)   while gewisseld do
(3)       boven := boven − 1;
(4)       gewisseld := false;
(5)       for j := 1 to boven do
(6)           if A[j] > A[j + 1] then
(7)               Verwissel(A[j], A[j + 1]);
(8)               gewisseld := true
(9)           fi
(10)      od
(11)  od
```

Bubble sort trivia

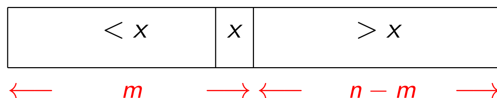
1. Bubble sort heeft best case complexiteit $O(n)$, average case complexiteit $O(n^2)$ en worst case complexiteit $O(n^2)$
2. In de praktijk is Insertion sort ongeveer 5 maal sneller dan Bubble sort
3. Hoewel het zowat het slechtst mogelijke sorteeralgoritme is, wordt IS veel gebruikt als simpele introductie
4. Bubble sort sorteert vrij efficient lijsten sorteert waar slechts 2 a 3 elementen op de verkeerde plaats staan

1. Verdeel de getallen in $\lfloor \frac{n}{5} \rfloor$ groepjes van 5 elementen[4] (en 1 groepje met de resterende n mod 5)
2. Vind de mediaan van elk van de $\lceil \frac{n}{5} \rceil$ groepjes van 5 (of minder), bijvoorbeeld met behulp van Bubblesort of opgave 24.
3. Vind de mediaan x van de in stap 2 gevonden $\lceil \frac{n}{5} \rceil$ medianen: recursie.

[4] deze 5 is niet zomaar een random gekozen waarde, maar is optimaal

Selectie is $O(n)$

4. Partitioneer (ongeveer zoals bij Quicksort) alle elementen rond x. Stel dat m getallen $\leq x$ zijn en $n - m$ getallen $> x$.



5. Vind de k-de in grootte uit m stuks als $k \leq m$, of de $(k - m)$-de uit $n - m$ als $k > m$: recursie.

De mediaan van ℓ elementen is de $\lceil \frac{\ell}{2} \rceil$-de in grootte.

Sorteren met Insertion Sort

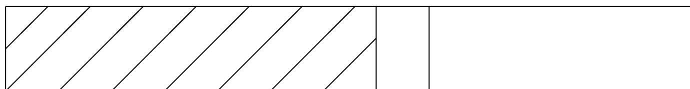
Probleem

Gegeven een rij (array) A met n elementen $A[1], \ldots, A[n]$. Sorteer A oplopend, dus $A[i] \leq A[i+1]$ voor alle i ($<$ als alle $A[i]$ verschillend zijn).

Sorteren met Insertion Sort

Insertion sort

i



$\longleftarrow$ al gesorteerd $\longrightarrow$

Conceptueel idee Insertion sort: itereer over i en voeg $A[i]$ op de juiste plek in het gesorteerde stuk $A[1] \cdots A[i-1]$ in door herhaald met de linkerbuur te vergelijken en te verwisselen indien nodig.

Sorteren met Insertion Sort

Het algoritme is gebaseerd op het doen van arrayvergelijkingen ($A[i] < A[j]$).

```
(1)   for i := 2 to n do
        // nu A[i] op de juiste plek in A[1] ... A[i − 1] invoegen
(2)       x := A[i];
(3)       j := i − 1;
(4)       while j > 0 and A[j] > x do
(5)           A[j + 1] := A[j];
(6)           j := j − 1;
(7)       od
(8)       A[j + 1] := x;
(9)   od
```

Sorteren met Insertion Sort

- Het aantal arrayvergelijkingen is een goede maat voor de complexiteit.
- Insertion sort doet eigenlijk steeds compare-exchange operaties: vergelijk en verwissel (indien nodig). Deze zijn hier vermomd als verschuivingen, waarna pas in de laatste stap $A[i]$ daadwerkelijk wordt neergezet.
- De verwisselingen zijn steeds buurverwisselingen.

Sorteren met Insertion Sort

We tellen het aantal vergelijkingen $A[j] > x$ $_{(= A[i])}$.

1. Worst case: $W(n) = \sum_{i=2}^{n}(i-1) = \frac{1}{2}n(n-1)$
2. Best case: $B(n) = \sum_{i=2}^{n} 1 = n-1$ Average case (*): $A(n) = \frac{1}{4}n(n-1) + n - \sum_{i=1}^{n} \frac{1}{i} \in \Theta(n^2)$

(*) onder de aanname dat alle $A[i]$'s verschillend zijn en dat alle n! permutaties (ordeningen) van $A[1]$ t/m $A[n]$ even waarschijnlijk zijn. We middelen dan over alle mogelijke permutaties en dat zijn in essentie alle mogelijke invoerrijtjes.

Sorteren met Insertion Sort

Definitie: een inversie van de permutatie $A[1], A[2], \ldots, A[n]$ is een paar $(A[i], A[j])$ waarvoor $i < j$ en $A[i] > A[j]$. M.a.w.: een inversie is een paar $(A[i], A[j])$ dat verkeerd om staat.

Merk op: *elk* sorteeralgoritme moet *alle* aanwezige inversies opheffen.

Verder: als een sorteeralgoritme altijd hooguit één inversie opheft per arrayvergelijking, dan is het aantal vergelijkingen dat wordt gedaan om $A[1], \ldots, A[n]$ te sorteren *ten minste* het aantal inversies van A.

Bovendien: een buurverwisseling (zoals bij Insertion sort) heft altijd precies één inversie op (indien de buurelementen verkeerd om staan).

Sorteren met Insertion Sort

Stelling

Het maximale aantal inversies dat kan voorkomen in een rijtje van n verschillende waarden is $\binom{n}{2} = \frac{1}{2}n(n-1)$. Dit treedt op bij een omgekeerd (= aflopend) gesorteerd rijtje.

Gevolg

Elk sorteeralgoritme (gebaseerd op arrayvergelijkingen) dat hooguit één inversie opheft per vergelijking doet ten minste $\frac{1}{2}n(n-1)$ vergelijkingen in de worst case.

Conclusie: Insertion sort is optimaal voor wat betreft de worst case, binnen de klasse van algoritmen gebaseerd op het doen van arrayvergelijkingen, waarbij per vergelijking hooguit één inversie wordt opgeheven (bijvoorbeeld via buurverwisselingen).

Sorteren met Insertion Sort

Stelling
Het *gemiddeld* aantal inversies in een permutatie van n verschillende waarden (bijvoorbeeld de getallen 1 t/m n) is $\frac{1}{4}n(n-1)$. Dit onder de aanname dat alle $n!$ permutaties even waarschijnlijk zijn.

Gevolg
Elk algoritme dat sorteert met behulp van arrayvergelijkingen en dat per vergelijking ten hoogste één inversie opheft, moet ten minste $\frac{1}{4}n(n-1)$ vergelijkingen doen in de average case.

Conclusie: Insertion sort doet gemiddeld $\frac{1}{4}n(n-1) + n - 1 - \sum_{i=2}^{n} \frac{1}{i}$ vergelijkingen, **dus** Insertion sort is in de average case in orde van grootte optimaal (binnen de betreffende klasse van algoritmen), namelijk $\Theta(n^2)$.

Sorteren met Insertion Sort

Voor sorteeralgoritmen gebaseerd op arrayvergelijkingen, waarbij per arrayvergelijking hooguit één inversie wordt opgeheven[5], geldt:

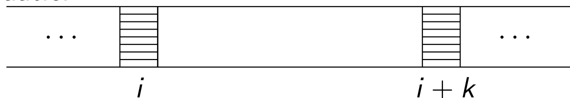
- # arrayvergelijkingen $\geq$ # inversies invoerarray
- # arrayvergelijkingen in de worst case $\geq \frac{1}{2}n(n-1)$

Als je een beter sorteeralgoritme (gebaseerd is op arrayvergelijkingen) wilt, moet je elementen verwisselen die verder van elkaar liggen, zoals Mergesort, Quicksort, Shellsort.

[5] dit is het geval bij algoritmen die gebruikmaken van buurverwisselingen, zoals Insertion sort en Bubblesort

Stel dat $A[i]$ en $A[i+k]$ $(k > 0)$ verkeerd om staan en dat we die verwisselen. Hoeveel inversies worden dan ten minste respectievelijk ten hoogste opgeheven?

Situatie:



met $A[i] > A[i+k]$. Verwissel nu $A[i]$ en $A[i+k]$.