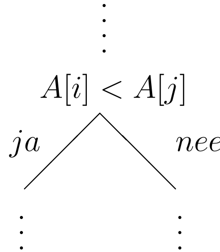
Uitgebreide uitwerking Tentamen Complexiteit, mei 2007

Opgave 1.

a. Een beslissingsboom beschrijft de werking van het betreffende algoritme (gebaseerd op arrayvergelijkingen) op elke mogelijke invoer. In de interne knopen staan de vergelijkingen die het algoritme doet:



De linkersubboom van een knoop beschrijft dan de verdere werking van het algoritme als $A[i] < A[j]$, de rechtersubboom als $A[i] > A[j]$. In de bladeren staat het eindantwoord: het algoritme stopt daar en is klaar. Een pad van de wortel naar een blad correspondeert met een executie (de achtereenvolgens gedane vergelijkingen) van het algoritme op een zekere invoer. De hoogte van de boom (wortel op niveau 0), zijnde de lengte van het langste pad, stelt dan precies het worst case aantal vergelijkingen voor dat het algoritme doet.

b. Er zijn $n(n-1)$ mogelijke combinaties $(A[i], A[j])$ met $i \neq j$ die het maximum en het minimum kunnen aangeven. Het algoritme moet werken voor elke willekeurige invoer, dus het moet elke (maximum, minimum)-combinatie kunnen vinden. Er zijn dus $n(n-1)$ mogelijke antwoorden, die alle gegeven moeten kunnen worden. Er moeten dus minstens zoveel bladeren zijn: $b = \#\text{bladeren} \geq n(n-1)$. Dit geldt voor elke beslissingsboom corresponderend met een algoritme voor dit probleem. Uit $h \geq \lceil \lg b \rceil$ volgt dan dat voor zulke beslissingsbomen $h \geq \lceil \lg n(n-1) \rceil = \Theta(\lg n)$, want $\lg n(n-1) = \lg n + \lg(n-1)$. Conclusie: het aantal vergelijkingen in de worst case voor *elk* algoritme dat het maximum en het minimum van n (verschillende) waarden bepaalt is $\Omega(n \lg n)$ (immers de afschatting gold voor *elke* beslissingsboom bij dit probleem).

c. Het maximale aantal inversies dat kan voorkomen is $\frac{1}{2}n(n-1)$. Immers er zijn maximaal $\binom{n}{2} = \frac{1}{2}n(n-1)$ verschillende paren, dus het aantal inversies is nooit meer dan $\frac{1}{2}n(n-1)$. Dit aantal kan ook bereikt worden: bij het aflopend gesorteerde rijtje staan immers alle mogelijke $\frac{1}{2}n(n-1)$ paren verkeerd om.

In het oplopend gesorteerde rijtje heb je nul inversies. In het ergste geval moet een sorteeralgoritme dus $\frac{1}{2}n(n-1)$ inversies opheffen. Elk algoritme dat A sorteert en dat per vergelijking altijd *hooguit* 1 inversie opheft, moet derhalve (in het ergste geval) *minstens* $\frac{1}{2}n(n-1)$ vergelijkingen doen om alle inversies op te heffen.

d. Bubblesort (en ook Insertion sort) vergelijkt buurelementen $A[i]$ en $A[i+1]$ en verwisselt ze indien nodig. Zo'n buursverwisseling heeft de eigenschap dat deze precies 1 inversie opheft (namelijk de inversie $(A[i], A[i+1])$). Dus Bubblesort heft per vergelijking hooguit 1 inversie op en voldoet dus aan de voorwaarde van de stelling. We mogen derhalve op grond van de stelling concluderen dat Bubblesort ten minste $\frac{1}{2}n(n-1)$ vergelijkingen doet in de worst case.

Quicksort vergelijkt (en verwisselt) verder uit elkaar gelegen elementen, waarbij meer dan 1 inversie per verwisseling kan worden opgeheven. De stelling mag daarom niet worden toegepast. Er is echter een ander eenvoudig argument dat de bewering bewijst. Neem namelijk het reeds gesorteerde rijtje. Daarop doet (de versie uit het dictaat van) Quicksort $n+1+n+\cdots+4+3 = n+1+n+\sum_{k=1}^{n-1} k \ -2-1 = \frac{1}{2}n(n-1)+2n-2 \geq \frac{1}{2}n(n-1)$ vergelijkingen. Het worst case aantal vergelijkingen is minstens zo groot als het aantal vergelijkingen voor een willekeurige invoer, dus is minstens $\frac{1}{2}n(n-1)$.

Opgave 2.

a. Het array wordt in twee stukken van elk $\frac{n}{2}$ elementen gesplitst, en van elk van beide stukken wordt weer op dezelfde manier (recursieve aanroepen) het aantal inversies bepaald. Dit levert de bijdrage $2\mathrm{S}(\frac{n}{2})$.
Wanneer het aantal inversies links en het aantal inversies rechts bekend is, moeten nog de paren elementen $(A[i], A[j])$ met $A[i]$ uit de linkerhelft en $A[j]$ uit de rechterhelft worden vergeleken. Dit zijn in totaal $\frac{n}{2}\cdot\frac{n}{2} = \frac{n^2}{4}$ paren, dus je hebt evenzoveel vergelijkingen nodig. Samen dus: $S(n) = 2S(\frac{n}{2}) + \frac{1}{4}n^2$.
Tenslotte het geval $n = 1$. Er zijn nu uiteraard geen paren om te vergelijken, dus $S(1) = 0$.
b. Eerst proberen we de oplossing te vinden door $S(n)$ herhaald in zichzelf in te vullen.

$S(n) = 2S(\frac{n}{2}) + \frac{1}{4}n^2 = 2\{2S(\frac{n}{2^2}) + \frac{1}{4}(\frac{n}{2})^2\} + \frac{1}{4}n^2 = 2^2S(\frac{n}{2^2}) + \frac{1}{8}n^2 + \frac{1}{4}n^2 = 2^2\{2S(\frac{n}{2^3}) + \frac{1}{4}(\frac{n}{2^2})^2\} + \frac{1}{8}n^2 + \frac{1}{4}n^2 = 2^3S(\frac{n}{2^3}) + \frac{1}{16}n^2 + \frac{1}{8}n^2 + \frac{1}{4}n^2 = \cdots =$(algemene vorm, vermoedelijk) $2^lS(\frac{n}{2^l}) + \frac{1}{2^{l+1}}n^2 + \frac{1}{2^l}n^2 + \cdots + \frac{1}{2^3}n^2 + \frac{1}{2^2}n^2 =$ (neem $l = k$) $2^kS(1) + n^2\{\frac{1}{2^{k+1}} + \frac{1}{2^k} + \cdots + \frac{1}{2^2}\}$ =(gebruik dat $S(1) = 0$) $\frac{n^2}{2}\{\frac{1}{2^k} + \cdots + \frac{1}{2}\}$ =(gebruik de hint) $\frac{n^2}{2}(1-\frac{1}{n}) = \frac{1}{2}n(n-1)$.

Nu nog bewijzen dat $S(n) = \frac{1}{2}n(n-1)$ inderdaad de oplossing is van de recurrente betrekking. Dit gaat m.b.v. volledige inductie.
(i) $S(n) = \frac{1}{2}n(n-1)$ voldoet inderdaad aan $S(1) = 0$.
(ii) Inductie-aanname: stel dat $S(n)$, de oplossing van de recurrente betrekking, gelijk is aan $\frac{1}{2}n(n-1)$ voor alle tweemachten n met $n < N$ en N een tweemacht ≥ 2. Dan moeten we laten zien dat dan ook $S(N) = \frac{1}{2}N(N-1)$.
Er geldt: $S(N) =$ (recurrente betrekking) $2S(\frac{N}{2}) + \frac{1}{4}N^2 =$ (inductie-aanname) $2\cdot\frac{1}{2}\cdot\frac{N}{2}(\frac{N}{2} - 1) + \frac{1}{4}N^2 = \frac{1}{4}N^2 - \frac{N}{2} + \frac{1}{4}N^2 = \frac{1}{2}N(N-1)$. QED

Opgave 3.

a. Alles (in orde van grootte) afschatten op regel 3 en/of regel 8 en/of regel 10.
#(regel 1) = 1
#(regel 2) = $\frac{n}{2} + 1$ (alleen even posities aflopen)
#(regel 3) = $\frac{n}{2}$
#(regel 4) $\leq$ #(regel 3) en #(regel 5) = #(regel 3)
Verder #(regel 3) ≥ 1 want $n - 1 \geq 1$, en dus #(regel 2) = O(#(regel 3))
#(regel 6) = 1
#(regel 7) = $\frac{n}{2} + 1$ (alleen oneven posities aflopen)
#(regel 8) = $\frac{n}{2}$
#(regel 12) = #(regel 9) en #(regel 9) $\leq$ #(regel 8)
Verder #(regel 8) ≥ 1 want $n - 1 \geq 1$, dus #(regel 7) = O(#(regel 8))
#(regel 13) = #(regel 8)
#(regel 11) $\leq$ #(regel 10)

Dus de tests in regel 3, 8 en 10 samen zijn inderdaad een goede maat voor de complexiteit. Bovendien zijn ze kenmerkend/bepalend voor dit algoritme: op grond van deze tests wordt steeds actie ondernomen.

b. Aangezien er evenveel positieve als negatieve elementen voorkomen moeten er evenveel paren $(A[i], A[i+1]) = (>0, >0)$ als paren $(A[i], A[i+1]) = (<0, <0)$ voorkomen (i oneven). Dat betekent dus dat als je zo'n paar van de vorm $(>0, >0)$ tegenkomt, je zeker weet dat er ook nog een paar $(<0, <0)$ voor moet komen. Merk op dat na afloop van het eerste gedeelte (regel 1 t/m 5) alle paren van de vorm $(A[i], A[i+1]) = (>0, <0)$ (i oneven) goedgezet zijn. Alleen paren $(A[i], A[i+1])$ (met i oneven) van de vorm $(<0, >0)$ (die staan dus goed), $(>0, >0)$ en $(<0, <0)$ komen dan dus nog voor. Als je een paar $(A[i], A[i+1])$ (i oneven) tegenkomt met $A[i] > 0$ weet je dus zeker dat je een paar $(>0, >0)$ te pakken hebt; een paar $(<0, <0)$ kun je eenduidig herkennen aan $A[j] < 0$ met j even. De while-lus uit regel 9 t/m 11 zoekt dus een paar $(<0, <0)$, terwijl je al een paar $(>0, >0)$ hebt gevonden. Aangezien je zeker weet dat er in dat geval nog zo'n paar is, hoef je in regel 10 niet te testen op $j \leq n$.

c. Merk op dat in deel 1 altijd $\frac{n}{2}$ vergelijkingen (regel 3) worden gedaan, onafhankelijk van de invoer. Ook de vergelijking in regel 8 gebeurt altijd $\frac{n}{2}$ keer, onafhankelijk van de invoer. Alleen het aantal keer dat regel 10 wordt uitgevoerd hangt af van de invoerrij.

Het beste geval hebben we als deze test nul keer wordt gedaan. Dit kan inderdaad voorkomen, namelijk als $A[i]$ voor oneven i altijd < 0 is in regel 8, m.a.w. als na fase 1 op alle oneven posities een negatief getal staat (en dus op alle even posities een positief getal). Dit is het geval voor alle invoerrijtjes waarbij alleen paren $(A[i], A[i+1])$ voorkomen van de vorm $(<, >0)$ of $(>0, <0)$. Het minimale aantal vergelijkingen is dan dus $\frac{n}{2} + \frac{n}{2} + 0 = n$.

d. Worst case als regel 10 juist zo vaak mogelijk wordt gedaan: dus voor zoveel mogelijk i's moet $A[i] > 0$ zijn in regel 8, en voor elke i zo ver mogelijk naar rechts lopen in de loop van regel 9 t/m 11. Het maximale aantal komt voor als er zoveel mogelijk paren $(>0, >0)$ zijn, en alle 'bijbehorende' paren $(<0, <0)$ rechts staan. Ergo: $\frac{n}{4}$ (n is een viervoud) paren $(>0, >0)$ staan allemaal links, en de $\frac{n}{4}$ paren $(<0, <0)$ allemaal helemaal rechts na fase 1. Dat betekent dat in de oorspronkelijke rij ook al alle $\frac{n}{2}$ positieve getallen geheel links staan, en daarachter alle $\frac{n}{2}$ negatieve getallen. Het aantal vergelijkingen is dan: $\frac{n}{2} + \frac{n}{2} + \frac{n}{4} + 1 + \frac{n}{4} + 2 + \frac{n}{4} + 3 + \cdots + \frac{n}{4} + \frac{n}{4} = n + \frac{n}{4} \cdot \frac{n}{4} + (1 + 2 + \cdots + \frac{n}{4}) = n + \frac{n^2}{16} + \frac{1}{2} \cdot \frac{n}{4}(\frac{n}{4} + 1) = \frac{3n^2}{32} + \frac{9n}{8}$.

e. In regel 9 wordt steeds helemaal links begonnen met zoeken naar de eerste even j waarvoor $A[j] < 0$ is. Een ordeverbetering is bijvoorbeeld eenvoudig te verkrijgen door bij te houden waar je in de vorige ronde (dus de vorige keer dat je een negatieve $A[j]$ zocht) gebleven was. Dat heeft tot gevolg dat je niet elke keer opnieuw een groot deel van het array doorloopt, wat in de worst case leidt tot een term die kwadratisch is in n, maar in totaal over alle rondes 1 keer door het array gaat: lineair dus!

Opgave 4.

a. We geven een *niet-deterministisch polynomiaal* algoritme A voor Part.

A heeft als invoer een eindige verzameling natuurlijke getallen S, hieronder ook wel genoteerd als x. (Het aantal elementen van S, zeg m, kan trouwens in $O(|S|)$ stappen worden bepaald, namelijk door S af te lopen.)

A doet het volgende:

1. Fase 1. Genereer een random string s, hierna te interpreteren als twee rijen gehele getallen, gescheiden door een ; (bijv.): O($|s|$). (We hopen dat s twee disjuncte deelverzamelingen voorstelt die samen S vormen, en die evengrote som hebben.)
2. Fase 2. Hier wordt gecontroleerd of s twee disjuncte deelverzamelingen van S voorstelt die samen S vormen, en evengrote som hebben.

* controleer dat elke $s_i \in s$ (behalve het scheidingsteken) in S zit: s aflopen en vergelijken met S. Dat kan in O($|s||x|$) stappen.
* controleer dat alle s_i verschillend zijn: O($|s|^2$).
* controleer dat het aantal getallen in s gelijk is aan m, het aantal elementen van S: O($|s|$) als we m al bepaald hebben.

Als deze drie controles positief zijn, dan betekent dat dat s inderdaad twee disjuncte deelverzamelingen voorstelt van S, die samen precies S opleveren.

* Controleer dat de som der elementen van de eerste deelverzameling gelijk is aan de som der elementen van de tweede: eerste stuk van s aflopen tot de ';' en alle s_i optellen, idem voor het tweede stuk. Dat kan in O($|s|$) stappen.

Als aan alle 4 eisen is voldaan stelt s een goede opsplitsing voor van S, en dan wordt True geretourneerd. Zodra een van de eisen niet klopt wordt False geretourneer of raak je in een oneindige lus.
3. Fase 3: uitvoerfase. Als Fase 2 True oplevert, dan wordt "ja" uitgevoerd, anders geen uitvoer.

Nu geldt: S is een ja-instantie is van Part $\Longleftrightarrow$ er bestaat een opsplitsing van S in twee disjuncte deelverzamelingen met gelijke som $\Longleftrightarrow$ er is een string s waarop A in Fase 2 True oplevert $\Longleftrightarrow$ er is een executie van A die "ja" oplevert $\Longleftrightarrow$ het antwoord van A op invoer S is "ja".
Kortom, A is inderdaad een algoritme voor Part. Bovendien is het polynomiaal. Immers voor ja-instanties is er een ja-executie waarin s een goede opsplitsing voorstelt. Dus dan is $|s| = |x|$. De executie met die s is dus O($|s|$) (Fase 1) + O($|x|^2$ + O($|x|^2$) + O($|x|$) + O($|x|$) (Fase 2) + O(1) (Fase 3) = O($|x|^2$).
Conclusie: A is een niet deterministisch polynomiaal (O($|x|^2$)) algoritme voor Part.
b. 1. Voor $S^* = \{s_1, s_2, \ldots, s_n, b, c\}$ geldt dat de som der elementen gelijk is aan $A + 2A - t + A + t = 4A$. Dus als S^* op te splitsen is in twee deelverzamelingen met gelijke som s^*, dan moet deze gelijk zijn aan de helft van $4A$, dus aan $2A$.
2. Merk op dat $b + c = 3A > s^*$, dus als S^* op te splitsen is in twee deelverzamelingen met gelijke som s^*, dan kunnen b en c nooit bij elkaar in dezelfde deelverzameling. Dus in zo'n geval zit b in de ene deelverzameling en c in de andere, beide aangevuld met elementen uit S.
Overigens volgt uit de grootte van b en c dat $b \neq s_i$ en $c \neq s_i$ voor alle $s_i \in S$.
c. "$\Longrightarrow$": Gegeven is dat S een deelverzameling S' bevat met $\sum_{s' \in S'} s' = t$. Bewering, dan kunnen we S^* opsplitsen in $\{b\} \cup S'$ en $\{c\} \cup S \setminus S'$. Dit zijn twee disjuncte deelverzamelingen van S^*, en samen vormen ze heel S^*. Verder geldt dat $b + \sum_{s' \in S'} s' = 2A - t + t = 2A$

en $c + \sum_{s' \in S \setminus S'} s' = A + t + A - t = 2A$. Dus dit is een goede opsplitsing, m.a.w. $S^* = T(< S, t >)$ is een ja-instantie van Part.
"$\Longleftarrow$": Stel dat S^* is op te splitsen in twee disjuncte deelverzamelingen S_1 en S_2, beide met totaalsom $2A$. Dan moet de ene verzameling, zeg S_1, b bevatten en de andere c. Dus $S_1 = \{b\} \cup S''$, waarbij S'' een deelverzameling van S, en dan $S_2 = \{c\} \cup S \setminus S''$. De som der elementen van S_1 is $2A$, dus $\sum_{s'' \in S''} s'' = 2A - b = t$. Dus S'' is een deelverzameling van S met totaalsom t, en derhalve is $< S, t >$ een ja-instantie voor SUM.
d. Merk op dat Part een bijzonder geval is van SUM, namelijk met $t = (\sum_{s \in S} s)/2$. Dit betekent dus dat we een eenvoudige polynomiale reductie van Part naar SUM hebben, namelijk S wordt afgebeeld op $< S, (\sum_{s \in S} s)/2 >$. Het is duidelijk dat dit een goede reductie is. (Immers S is op te splitsen in disjuncte S_1 en S_2 met gelijke som $(\sum_{s \in S} s)/2$) $\Longleftrightarrow$ S heeft een deelverzameling met som der elementen gelijk aan $(\sum_{s \in S} s)/2$). Ook triviaal dat de constructie polynomiaal is.) Dus Part $\leq_P$ SUM.
e. P is NP-hard als $Q \leq_P P$ voor alle Q uit $\mathcal{NP}$.
P is NP-volledig als (i) $P \in \mathcal{NP}$ en (ii) P is NP-hard.
f. Gegeven dat SUM NP-volledig is.
(i) Uit Part $\leq_P$ SUM en SUM NP-volledig volgt eigenlijk alleen maar (intuïtief) dat Part eenvoudiger is dan het heel moeilijke (namelijk NP-volledig) probleem SUM. Dat zegt verder nog niets over de moeilijkheidsgraad van Part. Part zou zelfs nog in $\mathcal{P}$ kunnen zitten. Dus bewering (i) is onwaar.
(ii) Uit SUM $\leq_P$ Part en SUM NP-volledig (en dus NP-hard) volgt dat ook Part NP-hard is (*). Immers $Q \leq_P$ SUM $\leq$ Part voor alle $Q \in \mathcal{NP}$, en uit de transitiviteit van $\leq_P$ volgt dan dat $Q \leq_P$ Part voor alle $Q \in NP$. Samen met het gegeven dat Part $\in \mathcal{NP}$ weten we dus dat Part NP-volledig is.

(*) Je kunt ook de volgende stelling uit het dictaat toepassen (komt op hetzelfde neer):
Stel P is een probleem waarvoor geldt dat $Q \leq_P P$ voor een of andere $Q \in \mathcal{NPC}$. Dan is P NP-hard. In deze opgave is $Q =$ SUM en $P =$ Part.