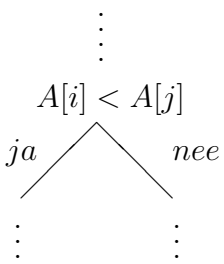
Uitwerking Tentamen Complexiteit, mei 2005

Opgave 1.

a. Elk algoritme (gebaseerd op arrayvergelijkingen) correspondeert met een binaire boom met in de interne knopen de vergelijkingen die het algoritme doet:



De linkersubboom van een knoop geeft de verdere werking van het algoritme als $A[i] < A[j]$, de rechtersubboom als $A[i] > A[j]$. Een pad van de wortel naar een blad correspondeert met een executie van het algoritme. De hoogte van de boom (wortel op niveau 0) stelt precies het worst case aantal vergelijkingen dat het algoritme doet voor. In de bladeren, waar het algoritme stopt, staat het eindantwoord.

b. Er zijn $n(n-1)$ mogelijke antwoorden: het aantal verschillende tweetallen (i, j) met $A[i]$ de grootste en $A[j]$ de op een na grootste. Deze mogelijkheden moeten allemaal kunnen voorkomen aangezien het algoritme voor elke invoerrij werkt. Dus het aantal bladeren: $b \geq n(n-1)$. Uit $h \geq \lceil \lg b \rceil$ volgt hier dat voor elke beslisingsboom $h \geq \lceil \lg n(n-1) \rceil$ $\geq \lg n(n-1) = \lg n + \lg(n-1)$, dus het aantal vergelijkingen in de worst case $= h = \Omega(\lg n)$ voor elk algoritme voor dit probleem.

c. Vergelijk de array-elementen twee aan twee ($\frac{n}{2} = 2^{k-1}$ vergelijkingen), vervolgens de $\frac{n}{2}$ winnaars weer twee aan twee ($\frac{n}{4} = 2^{k-2}$ vergelijkingen), de winnaars daarvan weer (2^{k-3} vergelijkingen), etcetera, totdat je na k rondes nog maar één array-element overhoudt: dit is dan de grootste van allemaal. Er zijn nu $2^{k-1} + 2^{k-2} + \ldots + 2 + 1 = 2^k - 1 = n - 1$ wedstrijden (vergelijkingen) en $k = \lg n$ rondes gespeeld.

De op een na grootste is alleen kleiner dan de grootste, dus kan van geen enkel array-element behalve de grootste verloren hebben. Hij moet dus een van de k verliezers van de grootste zijn, en wel de grootste van die k. Het kost nog eens $k-1 = \lg n - 1$ vergelijkingen om deze te bepalen. Het totaal aantal vergelijkingen is dus $n + \lg n - 2$.

d. Uit **b.** volgt slechts dat je *minstens* $\Omega(\lg n)$ vergelijkingen nodig hebt. Dit is slechts een ondergrens, die niet scherp hoeft te zijn. Je weet hieruit niet of er een algoritme met complexiteit $\Omega(\lg n)$ bestaat. Je mag dus niet uit **b.** concluderen dat de toernooimethode ($\Theta(n)$), niet optimaal is.

Opgave 2.

Het is voldoende te kijken naar een speciale invoer omdat worst case aantal vergelijkingen $\geq$ aantal vergelijkingen bijzonder geval. Kies de invoer zo dat de $\frac{n}{7}$ kleinste elementen op de plekken $7, 14, 21, \ldots, n$ staan, in oplopende volgorde. Bij alle rondes op de laatste (met stapgrootte 1) na blijven deze $\frac{n}{7}$ elementen op dezelfde plek staan omdat in die rondes alleen elementen vergeleken worden die een 7-voud van elkaar liggen. Pas bij het 1-sorteren worden deze $\frac{n}{7}$ elementen verplaatst, en wel naar hun uiteindelijke positie. Het element op plek 7 moet naar plek 1, die van plek 14 naar plek 2, etcetera. I.h.a. moeten ze allemaal

van plek $7i$ naar plek i. Dit alleen al kost $6+13+19+25+\ldots+6*i+1+\ldots+6*(\frac{n}{7})+1 \geq 6*(1+2+3+\ldots+i+\ldots+\frac{n}{7}) = 3*\frac{n}{7}*(\frac{n}{7}+1) \geq 3*\frac{n^2}{49}$ vergelijkingen. Dus in totaal worden voor deze invoer minstens $3*\frac{n^2}{49}$ vergelijkingen gedaan.

Opgave 3.

a. Hetzelfde sorteeralgoritme (recursie) wordt toegepast op de eerste $n/3$ elementen, de middelste $n/3$ en de laatste $n/3$. Dat is dus $3*S(n/3)$ vergelijkingen in de worst case. Vervolgens worden de drie rijtjes samengevoegd. In het ergste geval houd je zo lang mogelijk drie rijtjes (dan telkens 2 vergelijkingen nodig), totdat je twee rijtjes van elk 1 element overhoudt. Dan heb je $2*(n-2)$ vergelijkingen gedaan. Nu nog 1 om de grootste van de twee overgebleven waarden te bepalen, geeft samen $2(n-2)+1 = 2n-3$ vergelijkingen.
Beginwaarde: als $n = 1$ is de rij al gesorteerd, dus niets doen.

b. Probeer de oplossing te vinden door $S(n)$ herhaald in zichzelf in te vullen:
$S(n) = 3S(\frac{n}{3}) + 2n - 3 = 3*(3S(\frac{n}{3^2}) + 2\frac{n}{3} - 3) + 2n - 3 = 3^2S(\frac{n}{3^2}) + 2*2n - (3+3^2) = 3^2(3S(\frac{n}{3^3}) + 2\frac{n}{3^2} - 3) + 2*2n - (3+3^2) = 3^3S(\frac{n}{3^3}) + 3*2n - (3+3^2+3^3) = \ldots = 3^kS(\frac{n}{3^k}) + k*2n - (3+3^2+3^3+\ldots+3^k) = 3^kS(1) + k*2n - 3*(1+3+3^2+\ldots+3^{k-1}) = 2kn - \frac{3}{2}(3^k-1) = 2n*{}^3\log n - \frac{3}{2}(n-1)$. Gebruikt dat $k = {}^3\log n$.
Nu nog met inductie bewijzen dat $S(n) = 2n*{}^3\log n - \frac{3}{2}(n-1)$ (*) inderdaad de oplossing is van de recurrente betrekking.
Beginwaarde: $S(n) = 2n*{}^3\log n - \frac{3}{2}(n-1)$ voldoet aan $S(1) = 0$.
Inductieaanname: stel dat (*) geldt voor alle $n < N$ met n en N driemachten.
Er geldt: $S(N) =$ (recurrente betrekking) $3*S(N/3) + 2N - 3 =$ (inductie-aanname) $3*(2\frac{N}{3}*{}^3\log\frac{N}{3} - \frac{3}{2}(\frac{N}{3}-1)) + 2N - 3 = 2N({}^3\log N - 1) - \frac{9}{2}(\frac{N}{3}-1) + 2N - 3 = 2N*{}^3\log N - \frac{3}{2}N + \frac{3}{2} = 2N*{}^3\log N - \frac{3}{2}(N-1)$. Dus (*) geldt ook voor N. QED

Opgave 4.

a. Kenmerkend voor dit algoritme is het doen van optellingen; de rest is veelal boekhouding. Intuïtief: het meeste werk zal gebeuren in de binnenste while-lus, dus als maat voor de complexiteit kun je het aantal keer dat regel 5 wordt uitgevoerd nemen. Wat preciezer:
#(regel 1) = 1
#(test in regel 2) = $n+1$
#(regel 3) = n = #(regel 9) = #(regel 7)
#(regel 8) $\leq$ #(regel 7)
#(regel 6) = #(regel 5) #(volledige test regel 4) = #(regel 5) +1 per doorgang buitenste while, dus in totaal #(test regel 4) = #(regel 5) $+n$
Omdat $j = i \leq n$ en *hoeveel* $= 0 < t$ wordt regel 5 voor *elke* i ten minste 1 keer uitgevoerd, dus #(regel 5) $\geq n$. Dus #(regel 1, 3, 6, 7, 8, 9) $\leq$ #(regel 5). Verder volgt hieruit: #(regel 2) = O(#(regel 5)) en #(test regel 4) $\leq 2*$#(regel 5), dus #(regel 4) = O(#(regel 5)). Elke regel is dus in orde van groote af te schatten op regel 5.

b. Beste geval als voor elke i de binnenste while-lus meteen na de eerste doorgang stopt. Dan moet dus voor elke i *hoeveel* $\geq t$ worden direct bij de eerste stap, behalve eventueel als $i = n$. Dat is het geval als $A[i] \geq t$ voor elke i behalve eventueel voor $i = n$. Het aantal optellingen is dan n.

c. Worst case als de binnenste lus voor elke i zo ver mogelijk doorgaat. Voor alle $j \leq n$ moet dus steeds (voor elke j) hoeveel $< t$ zijn. Dan moet dus voor elke i gelden: $A[i] + \cdots + A[j] < t$ voor $j = i, \ldots, n-1$ (de waarde van $A[i] + \cdots + A[n]$ doet er niet toe). Dit is equivalent met de eis: $A[1] + \cdots + A[n-1] < t$. Het aantal optellingen is dan

$n + (n-1) + (n-2) + \cdots + 2 + 1 = \frac{1}{2}n(n+1)$.
d. In elke ronde wordt ofwel i ofwel j opgehoogd (tenzij je stopt). Worst case bijvoorbeeld als je eerst helemaal naar rechts loopt met j totdat $j = n$ (n vergelijkingen), en dan met i geheel naar rechts (ook n vergelijkingen). Dit gebeurt bijvoorbeeld als $A[1] + \cdots + A[n-1] < t$ en $A[n] \geq t$ (bijvoorbeeld het rijtje $1, 1, 1, \ldots, 1, t$). Het aantal optellingen is dan $n + n = 2n$.

Opgave 5.
a. Een *niet-deterministisch polynomiaal* algoritme A voor TSP+$\triangle$ is het volgende:
1. Fase 1. Genereer een random string s, hierna te interpreteren als een rij knopen uit V.
2. Fase 2. Controleert of s een Hamiltonkring voorstelt met totaalgewicht $\leq k$.
* controleer dat s bestaat uit precies n knopen: O($|s|$).
* controleer dat alle knopen uit s in V zitten: O($|s| \cdot |V|$)
* controleer dat alle knopen uit s verschillend zijn: O($|s|^2$)
* controleer dat het totaalgewicht van de Hamiltonkring s (als de eerste drie controles positief waren) $\leq k$ is: achtereenvolgende tweetallen knopen (takken) v_i, v_{i+1} uit s aflopen en de bijbehorende gewichten ophalen en optellen. Dat kan in O($|s|$) stappen als we de adjacency-matrix-representatie voor gewogen grafen gebruiken.
Als een van deze vier tests negatief uitvalt wordt door Fase 2 False teruggegeven (of oneindige loop), en anders True.
3. Uitvoerfase. Als Fase 2 True gaf, dan wordt ja uitgevoerd, anders geen uitvoer.
A heeft als invoer x een gewogen, ongerichte graaf $\mathcal{G} = (V, E)$ en een geheel getal $k \geq 0$. Er geldt: als x een ja-instantie is, dan is er een executie van A die ja oplevert. Voor die string s geldt dan dat $|s| \leq |V|$. De executie met die s is dus O($|V|^2$) = O($|x|^2$): polynomiaal in $|x|$. Verder, als x een nee-instantie is, dan is er geen string s die in Fase 2 wordt goedgekeurd, dus er is geen executie die ja oplevert. Conclusie: A is inderdaad een niet deterministisch polynomiaal algoritme voor TSP+$\triangle$.
b. Het volgende algoritme construeert $\mathcal{G}'$ (gebruik de adjacency-matrix-representatie) in een polynomiaal aantal stappen:
* maak een kopie van $\mathcal{G}$: O($|\mathcal{G}|$).
* alle tweetallen knopen (u, v) $(u \neq v)$ aflopen (dus alle matrixelementen bekijken, dus O($|\mathcal{G}|$). Als die tak in $\mathcal{G}$ voorkomt (controle O(1), dan wordt het een tak in $\mathcal{G}'$ met gewicht 1 (O(1), namelijk matrixelement op 1 zetten). Zo niet, dan wordt dat een tak met gewicht 2 (O(1)). Totaal dus O($|\mathcal{G}|$).
* $k = |V|$ nog bepalen. Dat betekent het aantal knopen tellen, dus ook O($|\mathcal{G}|$).
c. "$\Longrightarrow$": $\mathcal{G}$ heeft een Hamiltonkring $v_1, v_2, \ldots, v_n$ (dus met $n = |V|$ takken). Dan is dit (uiteraard) ook een Hamiltonkring in $\mathcal{G}'$. Bovendien bevat deze Hamiltonkring alleen takken die ook in $\mathcal{G}$ zitten, dus die hebben gewicht 1 in $\mathcal{G}'$. Zijn totaalgewicht $|V| * 1 = |V|$. Dus $\mathcal{G}'$ heeft een Hamiltonkring met totaalgewicht $\leq |V|$.
"$\Longleftarrow$": Zij $v_1, v_2, \ldots, v_n$ een Hamiltonkring in $\mathcal{G}'$ met totaalgewicht $\leq |V|$. Aangezien de kring $n = |V|$ takken bevat en $\mathcal{G}'$ alleen takken van gewicht 1 en 2 heeft moeten de gewichten van alle takken uit de kring wel 1 zijn. Met andere woorden: alle (v_i, v_{i+1}) en (v_n, v_0) zitten ook in $\mathcal{G}$. Dus $v_1, v_2, \ldots, v_n$ is ook een Hamiltonkring in $\mathcal{G}$.
d. 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.
e. Stelling: 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 (volgt uit transitiviteit van $\leq_P$). Als HC $\in \mathcal{NPC}$ dan kunnen

we de stelling toepassen met $Q = \text{HC}$ en $P = \text{TSP}+\triangle$. Verder is $\text{TSP}+\triangle \in \mathcal{NP}$, dus uit **a.**, **b.** en **c.** volgt dat TSP+$\triangle$ NP-volledig is: bewering (ii) is waar.
Uit $\text{TSP}+\triangle \in \mathcal{NPC}$ en $\text{HC} \leq_P \text{TSP}+\triangle$ volgt eigenlijk alleen maar (intuïtief) dat HC eenvoudiger is dan een heel moeilijk (NP-volledig) probleem. Dat zegt verder nog niets over de moeilijkheidsgraad van HC. Dus bewering (i) is onwaar.