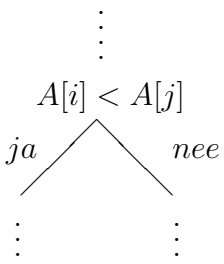
Uitwerking Tentamen Complexiteit, mei 2006

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. Een algoritme dat sorteert bepaalt in feite de permutatie/sortering/volgorde van de elementen die correspondeert met een oplopend rijtje. Het algoritme moet werken voor elke willekeurige invoer, dus moet elke permutatie kunnen vinden. Er zijn $n!$ mogelijke volgordes: $a_1 < a_2 < \ldots < a_n; a_2 < a_1 < \ldots < a_n; \ldots; a_n < a_{n-1} < \ldots < a_1$. Er zijn dus $n!$ mogelijke antwoorden, die alle gegeven moeten kunnen worden. Er moeten dus minstens zoveel bladeren zijn: $b = \#\text{bladeren} \geq n!$. 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
$h \geq \lceil \lg n! \rceil$, en dat is $\Theta(n \lg n)$. Ergo, het aantal vergelijkingen in de worst case voor *elk* algoritme dat de n waarden sorteert is $\Omega(n \lg n)$ (immers de afschatting gold voor *elke* beslissingsboom bij dit probleem).
c. Het is voldoende te kijken naar een speciale invoer en te bewijzen dat het aantal vergelijkingen daarvoor $\geq cn^2$ is voor zekere constante c (en vanaf zekere n_0). Immers: worst case aantal vergelijkingen $\geq$ aantal vergelijkingen bijzonder geval.
Kies de invoer zo dat de $\frac{n}{3}$ kleinste elementen op de plekken $3, 6, 9, \ldots, n$ staan, in oplopende volgorde. De posities van de overige waarden doen er niet toe. Bij het $\frac{n}{3}$-, $\frac{n}{9}$-, $\frac{n}{27}$-, $\ldots$, 9-, 3-sorteren worden er wel vergelijkingen gedaan, maar worden deze $\frac{n}{3}$ elementen niet verplaatst (de andere mogelijke wel), want er worden steeds alleen elementen vergeleken die een 3-voud van elkaar liggen. Dus na de op een na laatste ronde staan deze waarden nog steeds op dezelfde plek als bij het begin.
Pas bij het 1-sorteren gebeurt er iets mee en worden ze op hun juiste positie gezet. Ze moeten allemaal van plek $3i$ naar plek i. Dit alleen al kost $2 + 5 + 7 + 9 + \ldots + 2i + 1 + \ldots + 2 * (\frac{n}{3}) + 1 \geq 2 * (1 + 2 + 3 + \ldots + i + \ldots + \frac{n}{3}) = \frac{n}{3} * (\frac{n}{3} + 1) \geq \frac{n^2}{9}$ vergelijkingen. Dus in totaal worden voor deze invoer minstens $\frac{n^2}{9}$ vergelijkingen gedaan.

d. Eerst het aantal arrayvergelijkingen. Counting sort: 0; Shellsort (**c**): $\Omega(n^2)$; Merge sort: $\Theta(n \lg n)$; Binary Insertion sort: $\Theta(n \lg n)$; Shellsort Hibbard: $O(n^{\frac{3}{2}})$; Quicksort: $\Theta(n^2)$.
Counting sort is niet gebaseerd op het doen van arrayvergelijkingen, de complexiteit wordt bepaald door het aantal toekenningen aan array-elementen, en dat is $\Theta(n)$. Bij Binary Insertion sort is weliswaar het aantal vergelijkingen $\Theta(n \lg n)$, maar het aantal verschuivingen van array-elementen is $\Theta(n^2)$ in de worst case. Het aantal verschuivingen bepaalt hier dus de complexiteit. Voor de andere genoemde methoden is het aantal array-vergelijkingen wel een goede maat voor de complexiteit. In volgorde van worst case complexiteit hebben we dus: Counting sort, Merge sort, Shellsort (Hibbard), Quicksort, Binary Insertion sort, Shellsort (**c**).

Opgave 2.
a. Bepaal eerst de kleinste waarde door het array één keer door te lopen: $n-1$ vergelijkingen. Verwissel de kleinste met het achterste array-element. Zoek nu de grootste door het array tot en met het een na laatset element af te lopen: $n-2$ vergelijkingen. Verwissel de grootste met het voorste array-element. Totaal nodig hiervoor: $2n-3$ array-vergelijkingen. Nu staan het voorste en het achterste element al goed: resteren nog de $n-2$ overige elementen: $A[2]$ t/m $A[n-1]$. Deze sorteren we op dezelfde manier als het oorspronkeleijk array: recursie. Dat kost dus $S(n-2)$ vergelijkingen. We vinden dus: $S(n) = S(n-2) + 2n - 3$.
Als $n = 0$ valt er niets te sorteren, dus is het aantal vergelijkingen 0. (Ter controle: als $n = 2$ komt het algoritme neer op het vinden van de grootste (en kleinste) van twee elementen. Daarvoor is 1 vergelijking nodig. Ook volgens de recurrente betrekking geldt: $S(2) = S(0) + 1 = 1$.)
b. Eerst proberen we de oplossing te vinden door $S(n)$ herhaald in zichzelf in te vullen.
$S(n) = S(n-2)+2n-3 = S(n-4)+2(n-2)-3+2n-3 = S(n-4)+2(n-2)+2n-2*3 = S(n-6)+2(n-4)-3+2(n-2)+2n-2*3 = S(n-6)+2(n-4)+2(n-2)+2n-3*3 = \cdots =$ (algemene vorm) $S(n-h) + 2(n-h+2) + \cdots + 2(n-4) + 2(n-2) + 2n - \frac{h}{2} * 3 =$ (vul $h = n$ in) $S(0) + 2*(2 + \cdots + n - 2 + n) - \frac{n}{2} * 3 = 2*(2 + \cdots + n - 2 + n) - \frac{3n}{2} =$ (gebruik de hint) $2 * \frac{n}{2}(\frac{n}{2} + 1) - \frac{3n}{2} = \frac{1}{2}n(n-1)$.
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 even n met $n < N$ en N even ≥ 2. Dan moeten we laten zien dat dan ook $S(N) = \frac{1}{2}N(N-1)$.
Er geldt: $S(N) =$ (recurrente betrekking) $S(N-2) + 2N - 3 =$ (inductie-aanname) $\frac{1}{2}(N-2)(N-3) + 2N - 3 = \frac{1}{2}N^2 - \frac{5}{2}N + 3 + 2N - 3 = \frac{1}{2}N^2 - \frac{1}{2}N = \frac{1}{2}N(N-1)$. QED

Opgave 3.
a. Na ronde 3 (dus $i = 1, 2, 3$ gehad):
gedaan: T, T, T, F, F, F, F, F, F
kleiner: 0, 1, 8, 2, 2, 2, 2, 2, 2
groter: 8, 7, 0, 1, 1, 1, 1, 1, 1
Na ronde 5:
gedaan: T, T, T, T, T, F, F, F, F
kleiner: 0, 1, 8, 2, 3, 4, 4, 4, 4

groter: 8, 7, 0, 6, 5, 1, 1, 1, 1
Na ronde 6:
gedaan: T, T, T, T, T, T, F, F, F
kleiner: 0, 1, 8, 2, 3, 5, 5, 5, 4
groter: 8, 7, 0, 6, 5, 3, 1, 1, 2
Na ronde 7:
gedaan: T, T, T, T, T, T, F, F, F
kleiner: 0, 1, 8, 2, 3, 5, 5, 5, 4
groter: 8, 7, 0, 6, 5, 3, 1, 1, 2
Na ronde 8:
gedaan: T, T, T, T, T, T, F, F, F
kleiner: 0, 1, 8, 2, 3, 5, 5, 5, 4
groter: 8, 7, 0, 6, 5, 3, 1, 1, 2
Na ronde 9:
gedaan: T, T, T, T, T, T, F, F, T
kleiner: 0, 1, 8, 2, 3, 5, 6, 6, 4
groter: 8, 7, 0, 6, 5, 3, 1, 1, 4

b. #(regel 1) = 1 = #(regel 5) = #(regel 19)
#(regel 2) = $n + 1$
#(regel 3) = #(regel 4) = n
regel 5 t/m 18 is in feite een for-loop die ook stopt zodra *found* **True** wordt: maximaal aantal doorgangen is n
#(regel 6) $\leq n + 1$
#(regel 13), #(regel 14) $\leq$ #(regel 12) $\leq$ #(regel 11)
regel 9 t/m 15 is een for-loop
#(regel 16) = #(regel 18) = #(regel 8) = #(regel 9) = #(regel 6) $-1 \leq n$
#(regel 10) = $n + 1$ per doorgang buitenste while
#(regel 11) = #(regel 10) - 1 = n per doorgang buitenste while
Merk op dat regel 11 altijd minstens n keer wordt uitgevoerd. Immers: in de allereerste ronde ($i = 1$ en *found* is **False**) wordt de test in regel 11 al n keer gedaan. Gevolg:
#(regel 2), #(regel 6) $\leq n + 1 \leq$ #(regel 11) $+1 \leq 2$#(regel(11)
totaal #(regel 10) $\leq$ #(regel 11) $+n \leq 2$ #(regel(11)
Alle andere regels ook $\leq$ #(regel 11)
Elke regel is dus in orde van groote af te schatten op regel 11.

c. Merk op dat regels 7 t/m 18 zeker 1 keer gedaan worden, en (aangezien de test in regel 7 dan zeker **True** is) regel 11 in die eerste doorgang n keer. Indien $A[1]$ meteen de mediaan is wordt *found* daarna meteen **True** en stopt de buitenste while. Dus: best case als $A[1]$ de mediaan is en dan #(regel 11) = n en #(regel 12) = $n - 1$ (want niet voor $j = 1$).

d. Gegeven is dat $A[n]$ de mediaan is, dus *found* wordt pas in de laatste ronde **True**. Merk op dat de test in regel 7 voor $i = 1, 2, \ldots, \frac{n+1}{2}$ altijd **True** is, omdat er dan nog maar hooguit $\frac{n-1}{2}$ i's geweest zijn. Verder is $A[n]$ de mediaan, dus voor $i = n$ zal de test ook **True** zijn.
(i) Het beste geval zou dan zijn als voor de overige i de test in regel 7 steeds **False** oplevert. Dit kan ook echt voorkomen. Het aantal keer dat regel 11 wordt uitgevoerd is dan: $\frac{n+1}{2} \cdot n + n = \frac{n(n+3)}{2}$. Drie voorbeeldrijtjes: $1, 2, 3, 4, 6, 7, 8, 9, 5$; $2, 4, 3, 1, 6, 8, 9, 7, 5$ en $9, 7, 8, 6, 4, 2, 1, 3, 5$.

(ii) Het slechtste geval komt voor als de test in regel 7 altijd **True** oplevert; dan wordt regel 11 altijd (voor alle i) gedaan voor elke j ($= 1, \ldots n$). Dan is $\#(\text{regel } 11) = n \cdot n = n^2$. Drie voorbeeldrijtjes: $1, 9, 2, 8, 3, 7, 4, 6, 5$; $1, 2, 3, 4, 9, 8, 7, 6, 5$ en $8, 9, 6, 7, 1, 2, 3, 4, 5$.
e. Elk paar array-elementen wordt door het algoritme hooguit 1 keer bekeken (zie opgave). Het aantal array-vergelijkingen (regel 12) is dus maximaal $\binom{n}{2=\frac{1}{2}n(n-1)}$. Elk tweetal wordt dan precies één keer bekeken. Dit aantal wordt gehaald voor de in **d** gevonden worst case gevallen, dus als de test in regel 7 altijd **True** is. Voor elke i wordt $A[i]$ vergeleken met $A[i+1], A[i+2], \ldots, A[n]$. In totaal dus $n-1+n-2+\cdots+2+1 = \frac{1}{2}n(n-1)$ vergelijkingen.
f. Als het rijtje zo is dat regel 7 steeds **True** oplevert, en $A[n-1]$ is de mediaan, dan wordt $A[n]$ niet bekeken, maar die leverde in dat geval toch geen extra vergelijkingen op. Dus dit soort rijtjes zijn ook worst case gevallen.
Merk op: als de mediaan een van de eerste $n-2$ array-elementen is, wordt het worst case aantal vergelijkingen nooit gehaald.
Merk verder op dat als twee array-elementen $A[k]$ en $A[l]$ beide worden overgeslagen (regel 7 is **False**), dan worden ze nooit vergeleken. Dus dan wordt het worst case aantal niet gehaald. Echter, als er slechts één element $A[k]$ wordt overgeslagen, en $A[n]$ wordt ook bekeken (dus is de mediaan), dan worden alle in ronde k versmade vergelijkingen in de latere rondes alsnog gedaan. Kortom: ook dit is nog een worst case invoer.

Opgave 4.
a. We geven een *niet-deterministisch polynomiaal* algoritme A voor GHP. A heeft als invoer een gerichte graaf $\mathcal{G}$ met $\mathcal{G} = (V, E)$. Veronderstel dat de knopen genummerd zijn met 1 t/m n, waarbij $n = |V|$. (Het aantal knopen kan trouwens in $O(|\mathcal{G}|)$ stappen worden bepaald, namelijk door $\mathcal{G}$ af te lopen.)

A doet het volgende:
1. Fase 1. Genereer een random string s, hierna te interpreteren als een rij knopen uit V, dus als een rij gehele getallen: $O(|s|)$.
2. Fase 2. Hier wordt gecontroleerd of s een gericht Hamiltonpad voorstelt in $\mathcal{G}$.
* controleer dat s bestaat uit precies n integers (knoopnummers): s aflopen, tellen en vergelijken met n. Dat kan in $O(|s|)$ stappen.
* controleer dat alle integers tussen 1 en n zitten (wederom s aflopen), dus of de integers allemaal knopen uit V zijn: $O(|s|)$. (Opm. als je niet weet/aanneemt dat de knopen 1 t/m n genummerd zijn, loop dan voor elke integer uit s de knoopverzameling V af om te controleren of hij erin zit: $O(|s||V|) = O(|s||\mathcal{G}|)$.)
* controleer dat alle knopen uit s verschillend zijn: s aflopen en voor elke knoop die je tegenkomt de rest van s scannen. Dus: $O(|s|^2)$.
* controleer dat s een pad voorstelt, d.w.z. dat er tussen elk tweetal opeenvolgende knopen uit s een pijl zit. Dus: de achtereenvolgende tweetallen knopen v_i, v_{i+1} uit s aflopen ($O(|s|)$ en controleren of er een tak (=pijl) loopt van v_i naar v_{i+1} (bijv. bij adjacency-list in de buurlijst van v_i kijken of v_{i+1} daarin zit). Dit is $O(|s||E|) = O(|s||\mathcal{G}|)$.
(De eerste drie controles zijn een soort syntactische controle: er wordt gekeken of s inderdaad een verzameling van precies n verschillende knopen uit V voorstelt.) Als een van deze vier tests negatief uitvalt wordt (de rest niet meer gecontroleerd en 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.

Merk op dat geldt: als $\mathcal{G}$ een ja-instantie is van GHP, dan bestaat er een gericht Hamiltonpad in $\mathcal{G}$, dus dan is er een string s waarop Fase 2 True oplevert. M.a.w. als $\mathcal{G}$ een ja-instantie is, dan is er een executie van A die "ja" oplevert. Bovendien geldt voor die string s dan dat $|s| \leq |\mathcal{G}|$. De executie met die s is dus $O(|\mathcal{G}|) + O(|\mathcal{G}|^2) = O(|\mathcal{G}|^2)$ (het werk uit Fase 1 en Fase 2 opgeteld).
Anderzijds, als $\mathcal{G}$ een nee-instantie is bestaat er juist geen Hamiltonpad in $\mathcal{G}$, dus zal Fase 2 ook voor geen enkele s True opleveren, dus dan is er geen executie van A die "ja" oplevert.
Conclusie: A is een niet deterministisch polynomiaal ($O(|\mathcal{G}|^2)$) algoritme voor GHP.
b. Stel dat we de adjacency-matrix-representatie voor gerichte grafen gebruiken. Een algoritme dat de gegeven transformatie T uitvoert doet bijvoorbeeld het volgende:
* maak een adjacency-list met één knoop meer dan $\mathcal{G}$ heeft, waarbij de laatste knoop een lege buurlijst krijgt; dit is v'' (geen uitgaande takken). De buurlijsten van de overige knopen worden gewoon gekopieerd uit $\mathcal{G}$. De speciale knoop v heet nu v': uitgaande takken van v zijn uitgaande takken van v'. Totaal $O(|\mathcal{G}|)$.
* verander de uitgaande takken van de vorm (u, v) in pijlen van u naar v''. Dit kan door alle buurlijsten af te lopen en overal waar v staat deze in v'' te veranderen. Dit is eveneens $O(|\mathcal{G}|)$.
Bovenstaand algoritme construeert $T(\mathcal{G})$ dus $O(|\mathcal{G}|)$ stappen (polynomiaal).
c. "$\Longrightarrow$": $\mathcal{G}$ heeft een gerichte Hamiltonkring, zeg $v, v_2, \ldots, v_n$ (dus met $n = |V|$ takken). Dan is $v', v_2, \ldots, v_n, v''$ een Hamiltonpad in $\mathcal{G}'$. Immers: van v_i naar v_{i+1} gaat een pijl vanwege de Hamiltonkring in $\mathcal{G}$; verder volgt uit de constructie en de Hamiltonkring in $\mathcal{G}$ dat er een pijl loopt van v' naar v_2 (want van v naar v_2 in de Hkring), en van v_n naar v'' (want van v_n naar v in de Hkring).
"$\Longleftarrow$": $\mathcal{G}'$ heeft een gericht Hamiltonpad. Aangezien v' alleen uitgaande pijlen heeft en v'' alleen inkomende pijlen, moet v' de beginknoop van het Hamiltonpad zijn, en v'' de eindknoop. Het Hpad is dus van de vorm: $v', v^2, v^3, \ldots, v^n, v''$. Dan is $v, v^2, v^3, \ldots, v^n$ een gerichte Hamiltonkring in $\mathcal{G}$. Immers: van v^i naar v^{i+1} loopt een pijl vanwege het Hamiltonpad; van v naar v^2 gaat een pijl omdat (constructie van $\mathcal{G}$) er in het Hpad een pijl van v' naar v^2 loopt; analoog van v^n naar v (vanwege de pijl van v^n naar v'').
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. Als GHC $\in \mathcal{NPC}$ dan geldt in het bijzonder dat $Q \leq_P$ GHC voor alle $Q \in \mathcal{NP}$ (GHP is NP-hard). Uit **b** en **c** weten we dat GHC $\leq_P$ GHP. Uit de transitiviteit van $\leq_P$ volgt dan: $Q \leq_P$ GHC $\leq_P$ GHP $\rightarrow Q \leq_P$ GHP, voor alle $Q \in \mathcal{NP}$ (*). Dus GHP is NP-hard. Samen met **a** volgt hieruit dat GHP NP-volledig is. Dus bewering (i) is waar.
Uit GHP $\in \mathcal{NPC}$ en GHC $\leq_P$ GHP (en GHC $\in \mathcal{NP}$) volgt eigenlijk alleen maar (intuïtief) dat GHC eenvoudiger is dan een heel moeilijk (namelijk NP-volledig) probleem. Dat zegt verder nog niets over de moeilijkheidsgraad van GHC. GHC zou zelfs nog in $\mathcal{P}$ kunnen zitten. Dus bewering (ii) is onwaar.

(*) Dit komt eigenlijk neer op de volgende stelling uit het dictaat:
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 $Q =$ GHC en $P =$ GHP.