Uitgebreide uitwerking Tentamen Complexiteit, juni 2017

Opgave 1.

a. Een pad van de wortel naar een blad stelt de serie achtereenvolgende arrayvergelijkingen voor die het algoritme doet op zekere invoer. De hoogte van de beslissingsboom (lengte van het langste pad) stelt dan ook het aantal arrayvergelijkingen in de worst case voor. Voor zekere invoer volgt het algoritme zoals gezegd een pad in de boom; uiteindelijk stopt het algoritme voor die invoer in een blad; daar is het antwoord (voor de corresponderende invoer) dus bekend. Bladeren kunnen derhalve geassocieerd worden met de eindantwoorden/uitkomsten die het algoritme vindt.

b. Eerst twee belangrijke opmerkingen/stellingen over beslissingsbomen.

- Elk mogelijk antwoord moet als blad kunnen voorkomen omdat het algoritme voor elke mogelijke invoer van het probleem moet werken. Hieruit volgt onmiddellijk dat elke beslissingsboom (corresponderend met een algoritme voor het betreffende probleem) *minstens* zoveel bladeren heeft als er antwoorden mogelijk zijn, dus $b = \#\text{bladeren} \geq$ aantal mogelijke antwoorden. Leg deze $\geq$ in je antwoord ook uit.
- Verder hebben we een stelling voor binaire bomen die het verband aangeeft tussen de hoogte en het aantal bladeren b, namelijk $h \geq \lceil \lg b \rceil$. Noem deze stelling.

We moeten dus voor ons probleem het aantal mogelijke antwoorden, in dit geval de index van de grootste waarde en de index van de kleinste waarde (gr, 1nagr) voor arrays van de beschreven vorm, bepalen. Vanwege de speciale vorm van de invoerarrays, moet de grootste waarde altijd op een van de even posities staan: dat zijn $\frac{n}{2}$ mogelijkheden. De op een na grootste waarde kan dan nog op $\frac{n}{2}$ posities staan, namelijk het kan een van de $\frac{n}{2}$ resterende elementen op een even positie zijn, of de waarde die direct links van de grootste staat. Als voorbeeld: een mogelijk antwoord kan $(6, 2)$ zijn of $(4, 3)$, maar nooit $(6, 9)$. Immers $A[10]$ is in dit laatste geval zeker groter dan $A[9]$ (vanwege de speciale vorm van het array), maar natuurlijk ook $A[6]$, want dat was de grootste van het hele array. Daarom kan $A[9]$ nooit de op een na grootste zijn. In totaal heben we dus $\frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$ mogelijke antwoorden voor invoerarrays van het type uit de opgave. Dit betekent dat $b \geq \frac{n^2}{4}$.

Een en ander achter elkaar geschakeld: aantal arrayvergelijkingen in de worst case voor elk algoritme voor het probleem $= h \geq \lceil \lg b \rceil \geq \lceil \lg \frac{n^2}{4} \rceil = \lceil \lg n^2 - \lg 4 \rceil = \lceil 2 \lg n - 2 \rceil$.

c. We zoeken eerst de grootste waarde (even posities aflopen) en vervolgens de grootste van de linkerbuur van de grootste en de andere even posities. Dat levert in totaal $\frac{n}{2} - 1 + \frac{n}{2} - 1 = n - 2$ arrayvergelijkingen. Het algoritme:

```
gr = 2;
for (i=4; i<=n; i+=2) {
    if (A[i] > A[gr])
        gr = i;
} // n/2 - 1 arrayvergelijkingen
1nagr = gr - 1;
for (i=2; i<=n; i+=2) {
    if (i != gr) {
        if (A[i] > A[1nagr])
            1nagra = i;
}  // n/2 - 1 arrayvergelijkingen
```

d. (i) Dit is niet in tegenspraak met **b**. Daar is immers slechts bewezen dat elk algoritme voor ons probleem *minstens* $\lceil 2 \lg n - 2 \rceil$ arrayvergelijkingen doet in de worst case. Algoritme $\mathcal{B}$ kan zeker voldoen aan die eis, want $\Theta(\sqrt{n})$ is in $\Omega(\lg n)$, dus in orde van grootte echt meer dan $\lg n$. Bijvoorbeeld $2\sqrt{n}$ is echt groter dan $\lceil 2 \lg n - 2 \rceil$ voor alle $n \geq 2$.
(ii) Wel in tegenspraak met **b**. Immers, $O(\sqrt{\lg n})$ is altijd kleiner dan $\lceil 2 \lg n - 2 \rceil$ vanaf zekere n. En dat is in tegenspraak met het feit dat aantal arrayvergelijkingen (hier de maat voor de complexiteit) in het ergste geval altijd, dus voor alle n, minstens $\lceil 2 \lg n - 2 \rceil$ moet zijn.
(iii) Het algoritme van **c** doet altijd (dus ook in de worst case) $n - 2$ arrayvergelijkingen. Dit is echt meer dan de ondergrens, maar dat zegt niets over het al dan niet optimaal zijn van het algoritme. Die ondergrens hoeft namelijk helemaal niet scherp te zijn. Er hoeft dus geen algoritme te bestaan dat $\lceil 2 \lg n - 2 \rceil$ vergelijkingen doet. Het zou best kunnen dat het algoritme uit **c** toch optimaal is, en dat ons bewijs een te zwakke ondergrens heeft opgeleverd. Hoe dan ook, het is mogelijk dat **c** optimaal is, dus de bewering is niet in tegenspraak met **b**.
Ter illustratie: let nog eens op het probleem van het maximum van n getallen. Een beslissingsboomargument levert een ondergrens van $\lceil \lg n \rceil$, maar we hebben op een andere manier een betere ondergrens kunnen bewijzen: $n - 1$.

Opgave 2.
(i) Shellsort sorteert het array door achtereenvolgens te $\frac{n}{2}$-sorteren, $\frac{n}{2^2}$-sorteren, $\frac{n}{2^3}$-sorteren, ..., 2-sorteren, 1-sorteren. Bij ℓ-sorteren worden ℓ-deelrijtjes gesorteerd van elementen die telkens op afstand ℓ van elkaar liggen:
$A[1], A[1+\ell], A[1+2\ell], \ldots$
$A[2], A[2+\ell], A[2+2\ell], \ldots$
$\ldots$
$A[\ell], A[2\ell], A[3\ell], \ldots$
Het sorteren van deelrijtjes gebeurt met Insertion sort. De rij A wordt gegarandeerd gesorteerd omdat je eindigt met 1-sorteren, en dat is "gewoon" Insertion sort op het hele array.
(ii) Bij $\frac{n}{2^i}$-sorteren worden $\frac{n}{2^i}$ deelrijtjes ter lengte 2^i (namelijk n gedeeld door het aantal rijtjes $\frac{n}{2^i}$) gesorteerd met Insertion sort. Sorteren van een rijtje ter lengte 2^i is dus $O((2^i)^2)$ in de worst case. In totaal kost deze ronde dus $\frac{n}{2^i}$ keer $O((2^i)^2)$ arrayvergelijkingen, en dat is gelijk aan $O(n \cdot 2^i)$ arrayvergelijkingen (dus $\leq \text{Const} \cdot n \cdot 2^i$).
In totaal zijn er k rondes (immers $n = 2^k$). Dat betekent voor het totale aantal arrayvergelijkingen in de worst case dat we een en ander moeten sommeren. Dus: aantal arrayvergelijkingen in de worst case in totaal is $\leq \text{Const} \cdot \Sigma_{i=1}^{k} n \cdot 2^i \leq \text{Const} \cdot n \cdot \Sigma_{i=0}^{k} 2^i = \text{Const} \cdot n \cdot (2^{k+1} - 1) \leq \text{Const} \cdot n \cdot 2 \cdot 2^k = \text{Const} \cdot 2n^2 \in O(n^2)$.

Opgave 3.
a. We zorgen eerst dat het array in de juiste vorm (die uit opgave **1**) komt, dat wil zeggen, we moeten hebben dat $A[i] < A[i+1]$ voor oneven i. Dit doen we door voor alle oneven i $A[i]$ en $A[i+1]$ te vergelijken en indien nodig te verwisselen. Dit kost $\frac{n}{2}$ arrayvergelijkingen. Vervolgens gebruiken we het algoritme uit **1c** (dat is $n-2$ vergelijkingen) en zetten daarna de twee grootste waarden op hun juiste plek achteraan. In totaal zijn dit $\frac{n}{2} + n - 2 = \frac{3}{2}n - 2$ arrayvergelijkingen. Vervolgens doen we weer hetzelfde op het nog ongesorteerde voorstuk van $n - 2$ elementen. Dat is de recursieve aanroep, die $Z(n-2)$ vergelijkingen kost. Ten

slotte de beginconditie: als $n = 0$ valt er niets te sorteren, dus $Z(0) = 0$.
b. Herhaald invullen[1]:
$Z(n) = Z(n-2) + \frac{3}{2}n - 2 = Z(n-4) + \frac{3}{2}(n-2) - 2 + \frac{3}{2}n - 2 = Z(n-6) + \frac{3}{2}(n-4) - 2 + \frac{3}{2}(n-2) - 2 + \frac{3}{2}n - 2 = Z(n-6) + \frac{3}{2}(n-4+n-2+n) - 3\cdot 2 = Z(n-8) + \frac{3}{2}(n-6+n-4+n-2+n) - 4 \cdot 2 = \cdots =$ (vermoedelijke algemene vorm; klopt met de gevallen $\ell = 1, 2, 3, 4$)
$Z(n-2\ell) + \frac{3}{2}((n-2\ell+2) + (n-2\ell+4) + \cdots + n - 4 + n - 2 + n) - \ell \cdot 2 =$ (kies nu $\ell = k = \frac{n}{2}$, zodat we de beginwaarde $Z(0) = 0$ kunnen gebruiken; dan $n - 2\ell = n - 2k = 0$)
$= Z(0) + \frac{3}{2}(2+4+\cdots+n-4+n-2+n) - 2k = \frac{3}{2}\cdot 2\cdot(1+2+\cdots+\frac{n}{2}) - n = 3\cdot\frac{1}{2}\cdot\frac{n}{2}(\frac{n}{2}+1) - n = \frac{3}{8}n^2 + \frac{3}{4}n - n = \frac{3}{8}n^2 - \frac{1}{4}n$.

Je kunt de termen ook iets anders bij elkaar rapen, bijvoorbeeld (met de beginstappen overgeslagen): $Z(n) = Z(n-6) + \frac{3}{2}\cdot 3n - \frac{3}{2}(4+2) - 3\cdot 2 = Z(n-8) + \frac{3}{2}\cdot 4n - \frac{3}{2}(6+4+2) - 4\cdot 2 = Z(n-8) + \frac{3}{2}\cdot 4n - 3(3+2+1) - 4\cdot 2 =$ (algemene vorm) $Z(n-2\ell) + \frac{3}{2}\cdot\ell\cdot n - 3(\ell - 1 + \cdots + 3 + 2 + 1) - 2\ell$, en verder analoog aan hierboven.

Nu nog met volledige inductie bewijzen dat de gevonden $Z(n) = \frac{3}{8}n^2 - \frac{1}{4}n$, inderdaad de oplossing van de recurrente betrekking is voor *alle* n met n even en ≥ 0.

(i) $Z(n) = \frac{3}{8}n^2 - \frac{1}{4}n$ voldoet inderdaad aan $Z(0) = \frac{3}{8}0^2 - \frac{1}{4}0 = 0$, zien we door invullen.
(ii) Inductie-aanname: stel dat $Z(n)$, de oplossing van de recurrente betrekking, gelijk is aan $\frac{3}{8}n^2 - \frac{1}{4}n$ voor alle even $n \geq 0$ met $n < N$ en met N even. Dan moeten we laten zien dat de oplossing van de recurrente betrekking voor deze N (het volgende even getal dus) ook gelijk is aan $Z(N)$, dus gelijk is aan $\frac{3}{8}N^2 - \frac{1}{4}N$. Er geldt:
$Z(N) =$ (recurrente betrekking) $Z(N-2) + \frac{3}{2} - 2 =$ (inductie-aanname: formule geldt voor $N-2$) $\frac{3}{8}(N-2)^2 - \frac{1}{4}(N-2) + \frac{3}{2}N - 2 = \frac{3}{8}(N^2 - 4N + 4) - \frac{1}{4}N + \frac{1}{2} + \frac{3}{2}N - 2 = \frac{3}{8}N^2 - \frac{3}{2}N + \frac{3}{2} - \frac{1}{4}N + \frac{1}{2} + \frac{3}{2}N - 2 = \frac{3}{8}N^2 - \frac{1}{4}N$. QED.

Opgave 4.
a. Het kan voorkomen dat regel (7) nul keer wordt uitgevoerd, maar regel (4) (bijvoorbeeld) $\Theta(n)$ keer, een ordeverschil dus. Regel (7) alleen is dus zeker niet maatgevend voor de complexiteit, omdat er invoer bestaat waarvoor dit beslist niet het geval is. Dit gebeurt bijvoorbeeld als alle $A[i]$ verschillend zijn; dan is de test in regel (4) nooit True, dus regel (7) wordt 0 keer uitgevoerd, maar de test wordt wel $\frac{n}{2}$ keer gedaan. Dit is een ordeverschil, dus (7) alleen is niet maatgevend.
Overigens is regel (4) alleen ook geen goede maat voor de complexiteit; immers, elke keer dat de test in regel (4) True is wordt regel (7) n keer gedaan. Als bijvoorbeeld de major meteen in de eerste ronde wordt gevonden, gebeurt regel (4) 1 keer en regel (7) n keer en vervolgens stopt het algoritme. Wederom een ordeverschil, dus ook regel (4) alleen is niet maatgevend. Regel (4) en (7) samen blijken dat wel te zijn.
b. De bedoeling is om alle worst case invoer te vinden. Dat kan op een systematische manier door te redeneren vanuit het algoritme. Dit wordt ook aangegeven in de opgave.
In het ergste geval vinden zowel regel (4) als regel (7) zo vaak mogelijk plaats; dit is ook mogelijk. Regel (4) kan maximaal $\frac{n}{2}$ keer plaatsvinden, namelijk dan en slechts dan als `found` tot de laatste ronde False blijft. Dat betekent dat A geen major heeft of de major

[1]Merk op: $Z(n) = Z(n-2) + \frac{3}{2}n - 2$, dus $Z(n-2) = Z(n-2-2) + \frac{3}{2}(n-2) - 2 = Z(n-4) + \frac{3}{2}(n-2) - 2$ en evenzo $Z(n-4) = Z(n-6) + \frac{3}{2}(n-4) - 2$

M wordt in de laatste ronde gevonden. Dat laatste kan alleen als $A[n-1] = A[n] = M$. Regel (7) vindt daarbij zo vaak mogelijk plaats dan en slechts dan als voor elke bekeken i (dus alle oneven i) geldt dat $A[i] = A[i+1]$. We leiden hieruit af wat moet gelden opdat het invoerarray een worst case geval is.
We hebben gezien dat voor een worst case invoer moet gelden: $A[1] = A[2]$, $A[3] = A[4]$, $\ldots$, $A[n-1] = A[n]$ (*). Echter $A[1], A[3], \ldots, A[n-3]$ mogen geen major zijn, en hetzelfde geldt vanwege (*) dus voor $A[2], A[4], \ldots, A[n-2]$. Dat betekent dus dat $MA[n-1] = A[n]$ maar hooguit twee keer kan voorkomen, en dus geen major kan zijn. Het enige geval dat derhalve overblijft is dat A geen major heeft en dat (*) geldt. Dit zijn alle worst case gevallen.
Het aantal vergelijkingen in de worst case is nu $\frac{n}{2}$ keer regel (4) plus $\frac{n}{2} \cdot n$ regel (7). Samen $\frac{1}{2}n(n+1)$ arrayvergelijkingen.
Twee voorbeelden: $1, 1, 5, 5, 3, 3, 4, 4, 2, 2$ en $3, 3, 6, 6, 3, 3, 7, 7, 6, 6$.
c. We onderscheiden (i) het geval dat A geen major heeft en (ii) het geval dat A wel een major heeft.
(i) Als A geen major heeft blijft `found` tot en met de laatste ronde ($i = n-1$) False en dus wordt regel (4) dan zeker $\frac{n}{2}$ keer uitgevoerd. Het aantal arrayvergelijkingen is in dat geval minimaal als het aantal keer (7) minimaal is, te weten 0 keer. Dit komt voor dan en slechts dan als voor alle oneven i geldt dat $A[i] \neq A[i+1]$ In dat geval is het totaal aantal arrayvergelijkingen dus $\frac{n}{2}$, en alle best case gevallen zijn arrays waarvoor geldt dat $A[i] \neq A[i+1]$ voor alle oneven i. (Zulke arrays hebben automatisch geen major.)
(ii) Als A wel een major M heeft weten we zeker dat er een paar $A[i], A[i+1]$ is (i oneven) waarvoor $A[i] = A[i+1]$. Als we zo'n paar tegenkomen levert dat zeker 1 (regel (4)) + n (regel (7)) = $n+1$ arrayvergelijkingen op. Het gunstigst is als dit meteen gebeurt, dat wil zeggen als $A[1] = A[2] = M$. Immers, anders kost het minstens nog 1 of meer extra arrayvergelijkingen uit regel (4). Dus in het beste geval worden $n+1$ arrayvergelijkingen gedaan, en dit komt voor dan en slechts dan als geldt dat $A[1] = A[2] = M$. Waar de major in het array verder nog voorkomt maakt niet uit(, als hij nog maar op $\frac{n}{2} - 1$ andere posities staat). Voorbeeld: $7, 7, 4, 7, 9, 7, 7, 6, 7, 5$.

Opgave 5.
a. Eerst nog even wat algemene achtergrond. Voor NP-problemen geldt dat oplossingen polynomiaal te controleren zijn, maar niet polynomiaal te vinden (behalve als ze in $\mathcal{P}$ zitten). Dat betekent dat als iemand een oplossing gokt (vergelijk: Fase 1 hieronder) deze polynomiaal gecontroleerd kan worden (vergelijk Fase 2 hieronder). NP betekent zoals bekend Niet-deterministisch Polynomiaal, en daarom gieten we dit hele idee wat formeler/preciezer in de vorm van een niet-deterministisch algoritme zoals hieronder. Fase 1 en Fase 3 zien er altijd hetzelfde uit, Fase 2 is de controleerfase en is specifiek voor het onderhavige probleem. In dit geval gaat het om een ongerichte graaf waarvan we willen weten of deze een iso-3kleuring heeft. Het algoritme heeft een graaf $\mathcal{G}$ als invoer en moet voor deze graaf bepalen of het een ja-instantie is voor Iso-3Kleur is, dus of deze een iso-3kleuring heeft. We gokken derhalve een kleuring, waarvan we gaan controleren of dat een iso-3kleuring is. Merk op: je moet in de gok NIET (ook) een graaf gokken; die is immers al invoer voor je algoritme, en dus beschikbaar. Nu wat formeler, zoals in de opgave gevraagd.

We geven een *niet-deterministisch polynomiaal* algoritme A voor Iso-3Kleur. A heeft als invoer een ongerichte graaf $\mathcal{G} = (V, E)$ waarvan het aantal knopen m een drievoud is. We

mogen aannemen (zie de opgave) dat $V = \{1, 2, \ldots, m\}$.

A doet bijvoorbeeld het volgende:

1. Fase 1 (gokfase)
Er wordt een willekeurige string s gegenereerd.
// Deze s stelt hopelijk een iso-3kleuring voor van (de knopen) van $\mathcal{G}$;
// Dit wordt in fase 2 gecontroleerd.

2. Fase 2 (controlefase)
// Hier wordt gecontroleerd of s inderdaad een iso-3kleuring van de invoergraaf $\mathcal{G}$
// voorstelt. s moet in dat geval dus een rijtje van m kleuren zijn, zeg bestaan uit
// 1-en, 2-en en 3-en.
// Tevens moeten dan de twee gegeven eigenschappen gelden.

- controleer of s hooguit 3 verschillende kleuren bevat: s aflopen en checken dat elke s_i een waarde tussen 1 en 3 heeft. Dat kan in $O(|s|)$ stappen.
- controleer dat s precies m waardes bevat: s aflopen en tellen. Dat kan in $O(|s|)$ stappen.

Als aan deze voorwaarden is voldaan stelt s een kleuring van de knopen voor met maximaal 3 kleuren (als 1, 2, resp. 3 genummerd). Dit is dus een soort syntactische controle. Interpretatie van s: s_i, de i-de waarde uit s, stelt de kleur van knoop i voor.
Hierna volgende de twee meest essentiële controles. Geef van die dan ook wat explicieter aan hoe je die controles doet. Laten we aannemen dat we voor $\mathcal{G}$ de adjacency-list representatie gebruiken.

- controleer dat buurknopen een verschillende kleur hebben. Loop alle knopen af, en voor elke knoop de buurlijst. Dat is $O(|\mathcal{G}|)$. Haal voor elke knoop en zijn buurknoop (dus tak) in s hun kleur op (dat is $O(|s|)$ per tak) en controleer of die verschillen. Totaal is dat dan $O(|\mathcal{G}| \cdot |s|)$ stappen.

- controleer dat elke kleur even vaak (dus $\frac{m}{3}$ keer) voorkomt. s aflopen, drie tellers bijhouden en tellen: $O(|s|)$.

Als de vier controles positief zijn stelt s een iso-3kleuring van de knopen voor en retourneert het verificatie-algoritme (Fase 2) True. Zodra een van de controles niet klopt wordt False geretourneerd of raak je in een oneindige lus.

- Fase 3 (uitvoerfase)
Als Fase 2 True oplevert wordt "ja" uitgevoerd, anders geen uitvoer.

Nu geldt: Fase 2 geeft True $\Longleftrightarrow$ s is een iso-3kleuring van de knopen van $\mathcal{G}$. En dus: het antwoord van A op invoer $\mathcal{G}$ is "ja" (per definitie) $\Longleftrightarrow$ er is een executie van A die "ja" oplevert $\Longleftrightarrow$ er is een string s waarop A in Fase 2 True geeft $\Longleftrightarrow$ (ons concrete algoritme) er is een string s die een iso-3kleuring van de knopen van $\mathcal{G}$ voorstelt $\Longleftrightarrow$ $\mathcal{G}$ heeft een een iso-3kleuring $\Longleftrightarrow$ $\mathcal{G}$ is een ja-instantie van Iso-3Kleur.

Kortom, A is inderdaad een (niet-deterministisch) algoritme voor Iso-3Kleur. Bovendien is het polynomiaal. Immers, voor ja-executies stelt de gegokte s een kleuring van de knopen voor. In dat geval is dus $|s| \in O(|\mathcal{G}|)$. De (ja-)executie met die s kan derhalve zeker wel in: $O(|s|)$ (Fase 1) + $O(|s|)$ + $O(|s|)$ + $O(|\mathcal{G}| \cdot |s|)$ + $O(|s|)$ (Fase 2) + $O(1)$ (Fase 3) $\subseteq O(|\mathcal{G}|) + O(|\mathcal{G}|) + O(|\mathcal{G}|^2) + O(|\mathcal{G}|) \subseteq O(|\mathcal{G}|^2)$ stappen. Er is dus voor ja-instanties een ja-executie die $O(|\mathcal{G}|^2)$ stappen doet, en dat is polynomiaal in $|\mathcal{G}|$, de grootte van de invoer. Derhalve is A polynomiaal.

Conclusie: A is een niet deterministisch polynomiaal ($O(|\mathcal{G}|^2)$) algoritme voor Iso-3Kleur.

b. "$\Longrightarrow$": Stel $\mathcal{G}$ is een ja-instantie van 3Kleur. Dan bestaat er een correcte 3kleuring van de knopen van $\mathcal{G}$. Dan is de volgende kleuring een correcte iso-3kleuring van de knopen van $T(\mathcal{G})$.
Kleur de gekopieerde knopen v met dezelfde kleur als ze in de 3kleuring hadden. Geef bij elke knoop v de knopen v_1 en v_2 elk een van de twee resterende kleuren. Dus: als v kleur 3 heeft, dan kleuren we bijvoorbeeld v_2 met kleur 1 en v_1 met kleur 2. Op deze manier zijn hooguit drie kleuren gebruikt en hebben buurknopen nooit dezelfde kleur. Immers, de knopen v, die in de oorspronkelijke graaf zaten, waren daar correct gekleurd, en hier dus ook. De enige toegevoegde knopen zijn de v_1 en de v_2 behorend bij knoop v voor alle $v \in V$, waarmee ze een driehoek vormen. Die driehoekjes zijn correct gekleurd en hebben verder geen verbinding met de rest van de graaf dan via v. Dus buren hebben verschillende kleur. Tevens komt elke kleur precies even vaak, namelijk $|V|$ keer, want per driehoek komt elke kleur precies één keer voor.
"$\Longleftarrow$": Stel $T(\mathcal{G})$ is een ja-instantie van Iso-3Kleur. Dan is er dus een correcte iso-3kleuring van de knopen van $T(\mathcal{G})$. Daarvoor geldt in het bijzonder dat u en v waarvoor u en v in de oorspronkelijke graaf $\mathcal{G}$ zaten en $(u, v) \in E$ een verschillende kleur hebben. Dus als we de knopen van $\mathcal{G}$ kleuren met hun kleur uit de iso-3kleuring, dan hebben we een correcte 3kleuring van $\mathcal{G}$ te pakken.

c. P is NP-hard als $Q \leq_{\mathrm{P}} P$ voor alle Q uit $\mathcal{NP}$.
P is NP-volledig als (i) $P \in \mathcal{NP}$ en (ii) P is NP-hard.

d. (i) Gegeven is dat 2Kleur $\in \mathcal{P}$ en we weten dat $\mathcal{P} \subseteq \mathcal{NP}$. Dus 2Kleur $\in \mathcal{NP}$. Verder is 3Kleur $\in \mathcal{NPC}$, dus in het bijzonder geldt 3Kleur NP-hard is, en derhalve dat alle NP-problemen naar 3Kleur reduceren. Dat betekent dat ook 2Kleur $\leq_{\mathrm{P}}$ 3Kleur. Ook is 3Kleur $\leq_{\mathrm{P}}$ Iso-3Kleur, dus vanwege de transitiviteit van $\leq_{\mathrm{P}}$ volgt dat ook 2Kleur $\leq_{\mathrm{P}}$ Iso-3Kleur.
(ii) 3Kleur $\in \mathcal{NPC}$, dus in het bijzonder is 3Kleur NP-hard. Iso-3Kleur $\in \mathcal{NP}$, dus Iso-3Kleur $\leq_{\mathrm{P}}$ 3Kleur.
(iii) Het volgt uit **a** en (#). Immers, 3Kleur is NP-hard, dus voor alle $Q \in \mathcal{NP}$ geldt dat Q $\leq_{\mathrm{P}}$ 3Kleur en we weten ook dat 3Kleur $\leq_{\mathrm{P}}$ iso-3Kleur. Vanwege de transitiviteit van $\leq_{\mathrm{P}}$ geldt dan dat Q $\leq_{\mathrm{P}}$ Iso-3Kleur voor alle $Q \in \mathcal{NP}$. Ofwel: Iso-3Kleur is NP-hard. Samen met **a** bewijst dit dat Iso-3Kleur $\in \mathcal{NPC}$.
Uit Iso-3Kleur $\leq_{\mathrm{P}}$ 3Kleur kunnen we niet veel concluderen. Informeel staat hier dat Iso-3Kleur 'hooguit even moeilijk is als' het heel moeilijke 3Kleur. Hieruit kunnen we dus niets afleiden over de moeilijkheid van Iso-3Kleur. Dit probleem zou nog best in $\mathcal{P}$ kunnen zitten.

(iv) Stel dat het bewijs klopt en dat Iso-3Kleur $\leq_{\mathrm{P}}$ 2Kleur. Aangezien 2Kleur $\in \mathcal{P}$ en we uit (iii) weten dat Iso-3Kleur $\in \mathcal{NPC}$, hebben we een NP-volledig probleem polynomiaal gereduceerd naar een probleem uit $\mathcal{P}$. Volgens een stelling van college heeft dat tot gevolg dat dan $\mathcal{P} = \mathcal{NP}$. (Het bewijs van die stelling gebruikt de transitiviteit van $\leq_{\mathrm{P}}$ om aan te tonen dat $\mathcal{NP} \subseteq \mathcal{P}$. Samen met $\mathcal{P} \subseteq \mathcal{NP}$ levert dit op dat $\mathcal{NP} = \mathcal{P}$.) Vele wetenschappers hebben al geprobeerd dit te bewijzen, maar dit is nooit gelukt. Algemeen wordt aangenomen dat $\mathcal{P} \neq \mathcal{NP}$ geldt, hoewel ook dat nog niemand heeft kunnen bewijzen. Hoe dan ook, het is dus zeer onwaarschijnlijk dat het bewijs van je neef correct is.