Uitgebreide uitwerking Tentamen Complexiteit, juni 2016

Opgave 1. (3+10+4+7+6)
a. De hoogte van de beslissingsboom (lengte van het langste pad) stelt het aantal array-vergelijkingen in de worst case voor. Voor zekere invoer volgt het algoritme 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 antwoorden die het algoritme vindt.

b. Elk algoritme voor het bekeken probleem (het vinden van de mediaan, de middelste in grootte, voor het speciale soort rijtjes zoals in de opgave gedefinieerd (of de index van de mediaan)) moet voor elk van de mogelijke rijtjes het juiste antwoord (= (index van) de mediaan) kunnen vinden. Voor de middelste in grootte van een oneven aantal verschillende waardes n geldt dat er precies $\frac{n-1}{2}$ kleiner zijn en ook precies $\frac{n-1}{2}$ groter.
Vanwege de speciale vorm van de rijtjes kan niet elk array-element de middelste in grootte zijn. Deze rijtjes bevatten immers precies $\lceil \frac{n}{2} \rceil = \frac{n+1}{2}$ negatieve getallen en $\lfloor \frac{n}{2} \rfloor = \frac{n-1}{2}$ positieve getallen. De positieve getallen kunnen nooit de mediaan zijn, aangezien er altijd al ten minste $\frac{n+1}{2}$ array-elementen zeker kleiner zijn (namelijk de negatieve getallen). De negatieve array-elementen zouden allemaal wel de mediaan kunnen zijn. Het totaal aantal mogelijke antwoorden is dus $\frac{n+1}{2}$, namelijk alle $A[i]$ met i oneven.
Aangezien voor de gegeven soort invoerrijtjes de mediaan op al deze posities kan voorkomen, en elk algoritme voor al zulke invoerrijtjes moet werken, moet elke beslissingsboom (corresponderend met een algoritme voor het probleem) *minstens* zoveel bladeren hebben als er antwoorden mogelijk zijn: $b = \#\text{bladeren} \geq \frac{n+1}{2}$. Dit geldt dus voor elke beslissingsboom corresponderend met een algoritme voor dit probleem. Uit $h \geq \lceil \lg b \rceil$ (eigenschap van binaire bomen) volgt dan dat voor zulke beslissingsbomen $h \geq \lceil \lg \frac{n+1}{2} \rceil = \lceil \lg(n+1) - \lg 2 \rceil = \lceil \lg(n+1) - 1 \rceil$. Ergo, het aantal vergelijkingen in de worst case (dat is namelijk de hoogte van de bijbehorende beslissingsboom, zie **a**) voor *elk* algoritme voor het onderhavige probleem is $\lceil \lg(n+1) - 1 \rceil = \lceil \lg(n+1) \rceil - 1$ (immers de afschatting gold voor *elke* beslissingsboom bij dit probleem).

c. Voor de mediaan geldt in dit geval (n oneven) dat er $\frac{n-1}{2}$ waarden groter moeten zijn en $\frac{n-1}{2}$ kleiner. Er zijn $\frac{n-1}{2}$ positieve getallen en $\frac{n+1}{2}$ (dat is dus 1 meer) negatieve. De mediaan is dus een van de negatieve getallen. We weten zelfs, aangezien er in elk geval $\frac{n-1}{2}$ getallen, namelijk de positieve getallen, groter zijn dan elk negatief getal, dat de mediaan de grootste van de negatieve getallen moet zijn. We kunnen de middelste in grootte (= grootste van de negatieve getallen) dus vinden door de oneven posities af te lopen en de standaardmethode voor het vinden van het maximum te gebruiken. Het aantal vergelijkingen is dan $\lceil \frac{n}{2} \rceil - 1 = \frac{n-1}{2}$ (voor elke invoer van de aangegeven vorm)

Algoritme:

```
mediaan = 1;
for (i=3; i <= n; i+=2)
    if (A[i] > A[mediaan])
        mediaan = i;
```

d. *Stelling:* elk algoritme gebaseerd op arrayvergelijkingen dat het maximum van n verschillende waarden $A[1], A[2], \ldots, A[n]$ doet in de worst case ten minste $n-1$ arrayvergelijkingen.
Bewijs uit het ongerijmde: Stel het is niet waar. Dan is er dus een algoritme dat in de worst case echt minder dan $n-1$ arrayvergelijkingen doet; dat algoritme doet dus minder dan $n-1$ vergelijkingen voor elke invoer (*).
Merk nu op dat er precies 1 element de grootste is (want alle waarden zijn verschillend), en precies $n-1$ niet de grootste. Verder: na elke arrayvergelijking weten we van precies 1 van beide array-elementen zeker dat die niet het maximum is, namelijk van de kleinste van de twee. Stel dat het algoritme $n-2$ of minder vergelijkingen doet; dan weten we maar van hooguit $n-2$ array-elementen zeker dat die niet het maximum zijn. Er zijn dan nog minstens 2 elementen over die allebei het maximum kunnen zijn. Het algoritme kan in dat geval dus nooit zeker weten wat het maximum is. Tegenspraak met de juistheid van het algoritme.
De aanname (*) is dus niet waar. Ergo: de bewering van de stelling is wel waar. QED.
Zie ook dictaat pagina 4.
(Eigenlijk hebben we hierboven bewezen dat elk algoritme *altijd* ten minste $n-1$ arrayvergelijkingen moet doen.)

e. (i) Het algoritme uit **c** doet $\frac{n-1}{2}$ arrayvergelijkingen. In **b** hebben we een ondergrens van $\lceil \lg(n+1) - 1 \rceil$ afgeleid. Er is dus een echt ordeverschil tussen de ondergrens en de complexiteit van ons algoritme. We kunnen hier nog niets uit concluderen; het algoritme kan optimaal zijn (en de ondergrens dan dus niet scherp; misschien kunnen we met wat meer moeite een wel een ondergrens van $\frac{n-1}{2}$ bewijzen), maar het kan ook zijn dat we gewoon een slecht algoritme hebben bedacht en dat veel het beter kan. Misschien kan de ondergrens zelfs wel gehaald worden. We kunnen dus noch concluderen dat het algoritme niet optimaal is noch dat het wel optimaal is.
(ii) Merk op dat het vinden van de mediaan in arrays van de bekeken vorm precies neerkomt op het vinden van het maximum van de $\lceil \frac{n}{2} \rceil$ negatieve getallen. Zie ook **c**. Kortom, het probleem dat we moeten oplossen is het vinden van het maximum van $\lceil \frac{n}{2} \rceil$ waarden. Uit **d** volgt daarvoor een ondergrens van $\lceil \frac{n}{2} \rceil - 1$ arrayvergelijkingen in de worst case. Dat is precies het aantal vergelijkingen dat het algoritme uit **c** doet. Dit algoritme is derhalve optimaal (binnen de klasse van algoritmen gebaseerd op arrayvergelijkingen en opde bekeken klasse van invoerarrays).

Opgave 2. (5+10)
a. $W(n)$ = aantal verwisselingen van array-elementen dat het beschreven recursieve algoritme doet op een array met afwisselend positieve en negatieve getallen (zie de opgave). Om het hele array te reorganiseren zoals aangegeven worden eerst de linkerhelft en de rechterhelft op dezelfde manier rereorganiseerd. Beide helften hebben dezelfde vorm als heel A. Het probleem voor n is dus teruggebracht tot twee keer het probleem voor $n/2$; vandaar de term $2W(\frac{n}{2})$.
Vervolgens hebben we een array van de volgende vorm: $n/4$ negatieve getallen gevolgd door $n/4$ positieve getallen, dan weer $n/4$ negatieve getallen en ten slotte $n/4$ positieve getallen. Voorbeeld: `-5 -12 -7 -23 8 10 19 15 -26 -13 -20 -4 17 5 22 11`.
Nu moeten we de $n/4$ negatieve getallen uit de tweede helft en de $n/4$ positieve getallen uit de eerste helft nog onderling verwisselen, bijvoorbeeld door de twee middel-

ste array-elementen —$A[m]$ en $A[m+1]$— te verwisselen en vervolgens in de linkerhelft naar links te lopen en in de rechterhelft naar rechts en steeds de corresponderende elementen te verwisselen. (Als het array loopt van index `links` t/m index `rechts` is het midden $m = \lfloor(\texttt{links} + \texttt{rechts})/2\rfloor$.) Voor het voorbeeld krijgen we dan het array `-5 -12 -7 -23 -4 -20 -13 -26 15 19 10 8 17 5 22 11`. Hoe dan ook, we kunnen dit op de geschetste manier doen met behulp van $\frac{n}{4}$ verwisselingen, hetgeen de term $\frac{n}{4}$ verklaart.
Ten slotte het basisgeval. Als $n = 2$ ziet het array er zo uit: `>0 <0`. In dit geval moeten we het eerste en het tweede array-element verwisselen om de juiste vorm te krijgen. Dus het aantal verwisselingen is 1, ofwel: $W(2) = 1$.

b. Eerst proberen we de oplossing te vinden door $W(n)$ herhaald in zichzelf in te vullen. Merk op: $W(n) = 2W(\frac{n}{2}) + \frac{n}{4}$, dus $W(n/2) = 2W(\frac{n/2}{2}) + \frac{n/2}{4} = 2W(\frac{n}{2^2}) + \frac{n}{8}$ en $W(n/2^2) = 2W(\frac{n/2^2}{2}) + \frac{n/2^2}{4} = 2W(\frac{n}{2^3}) + \frac{n}{16}$. Dus:

$$W(n) = 2W(\frac{n}{2}) + \frac{n}{4} = 2\{2W(\frac{n}{2^2}) + \frac{n}{8}\} + \frac{n}{4} =$$

$$2^2W(\frac{n}{2^2}) + \frac{n}{4} + \frac{n}{4} = 2^2W(\frac{n}{2^2}) + 2 \cdot \frac{n}{4} =$$

$$2^2\{2W(\frac{n}{2^3}) + \frac{n}{16}\} + 2 \cdot \frac{n}{4} = 2^3W(\frac{n}{2^3}) + \frac{n}{4} + 2 \cdot \frac{n}{4} =$$

$$2^3W(\frac{n}{2^3}) + 3 \cdot \frac{n}{4}$$

De algemene vorm is vermoedelijk:

$$W(n) = 2^\ell W(\frac{n}{2^\ell}) + \ell \cdot \frac{n}{4}$$

Kies nu $\ell = k - 1$, zodat we de beginconditie $W(2) = 1$ kunnen gebruiken. Immers als $\ell = k - 1$ geldt, omdat $n = 2^k$, dat $\frac{n}{2^\ell} = \frac{2^k}{2^{k-1}} = 2$. Dan vinden we (gebruik de hint):

$$W(n) = 2^{k-1}W(2) + (k-1) \cdot \frac{n}{4} = 2^{k-1} + (k-1) \cdot \frac{n}{4} = \frac{n}{2} + (\lg n - 1) \cdot \frac{n}{4} =$$

$$\frac{n}{4} + \frac{n}{4}\lg n = \frac{n}{4}(1 + \lg n)$$

Nu nog bewijzen dat de gevonden $W(n) = \frac{n}{4}(1 + \lg n)$ inderdaad de oplossing is van de recurrente betrekking. Dit gaat m.b.v. volledige inductie.

(i) $W(n) = \frac{n}{4}(1 + \lg n)$ voldoet inderdaad aan $W(2) = \frac{2}{4}(1 + \lg 2) = \frac{1}{2}(1 + 1) = 1$, zien we door invullen.
(ii) Inductie-aanname: stel dat $W(n)$, de oplossing van de recurrente betrekking, gelijk is aan $\frac{n}{4}(1 + \lg n)$ voor alle 2-machten $n \geq 2$ met $n < N$ en met N ook een 2-macht. Dan moeten we laten zien dat dit ook geldt voor de oplossing van de recurrente betrekking voor deze N (de volgende 2-macht dus) ofwel we moeten aantonen dat $W(N) = \frac{N}{4}(1 + \lg N)$ als dat voor kleinere tweemachten ook geldt.

Er geldt:
$W(N)$ = (recurrente betrekking) $2W(\frac{N}{2}) + \frac{N}{4}$ = (inductie-aanname: formule geldt voor $N/2$) $2 \cdot \{\frac{N/2}{4}(1 + \lg(N/2))\} + \frac{N}{4} = \frac{N}{4}(1 + \lg N - 1) + \frac{N}{4} = \frac{N}{4} \lg N + \frac{N}{4} = \frac{N}{4}(1 + \lg N)$
QED.

Opgave 3. (4+8+7+5)
a. De index i loopt van 1 tot n en i wordt steeds met 1 of meer opgehoogd: zie regel (13), waar i gelijk wordt aan $k + 1$ en $k \geq i$. Dit laatste volgt meteen uit het algoritme (regel (3), (7) en (10)). Er wordt alleen gestopt als i groter of gelijk is aan n. De binnenste while-loop (met j) is in feite een for-lus: voor alle $j = i + 1, i + 2, \ldots, n$ wordt gekeken of $A[j] < A[i]$. Indien dat het geval is wordt $A[j]$ met $A[i]$ verwisseld; als $A[j] \geq A[i]$ wordt $A[i]$ niet veranderd. Op positie i staat dus direct voor regel (12) steeds de kleinste van de elementen $A[i]$ t/m $A[j]$. Na de binnenste while bevat $A[i]$ dus de kleinste van het staartstuk $A[i]$ t/m $A[n]$. Verder: als we een echt kleiner element tegenkomen wordt dit naar plek i verwisseld en wordt k weer gelijk gemaakt aan i; als we vervolgens een $A[j]$ aantreffen die gelijk is aan $A[i]$ blijft $A[i]$ onveranderd (dus de huidige kleinste van het staartstuk), maar wordt k tot $i + 1$ opgehoogd en wordt dat gelijke element verwisseld met deze $A[i + 1]$. Ook volgende gelijke elementen worden op deze manier op de posities $i + 2, i + 3, \ldots$ gezet, totdat we eventueel weer een echt kleinere waarde dan de huidige $A[i]$ tegenkomen. Enzovoort.
Conclusie: direct na de binnenste while geldt dat $A[i]$ t/m $A[k]$ gelijk zijn, en wel aan de kleinste waarde van het staartstuk $A[i], \ldots, A[n]$. De rest van het array (posities $k + 1$ t/m n) bevat waardes groter dan $A[i]$. Alles voor $A[i]$ is uiteraard kleiner dan $A[i]$.
De elementen $A[1]$ t/m $A[k]$ staan allemaal al goed, en we kunnen meteen doorgaan met positie $k+1$, op dezelfde manier. Dit is dus een versie van Selection sort waarbij we gelijke elementen ook meteen vooraan zetten. Hierdoor kunnen we soms rondes overslaan.

b. De binnenste while-loop is een for-lus, en doet voor vaste i precies $n - i$ (namelijk voor $j = i + 1, \ldots, n$) arrayvergelijkingen in regel (5). Het totale aantal arrayvergelijkingen hangt af van hoeveel (en welke) rondes je moet doen. Er zijn maximaal $n - 1$ rondes mogelijk, want i start op 1 en loopt tot en met $n - 1$. Het maximale aantal arrayvergelijkingen krijg je als je al die rondes ook echt moet doen. Dat is het geval d.e.s.d.a. (dan en slechts dan als) voor elke $i < n - 1$ in regel (13) i met precies 1 wordt opgehoogd, dus als k in elk van deze rondes eindig op i. In dat geval worden de rondes 1 t/m $n - 1$ allemaal gedaan. Dat betekent dat de uiteindelijke $A[i]$ (dus de waarde die na de binnenste while op positie i staat; die waarde is de kleinste van het staartstuk) maar één keer in A voorkomt. In dat geval wordt na ronde i steeds ronde $i + 1$ gedaan en wordt derhalve geen enkele ronde overgeslagen. Dit geldt voor elke $i = 1, \ldots, n - 2$. Na de laatste ronde ($i = n - 1$) wordt sowieso gestopt en maakt het dus niet uit of k voor $i = n - 1$ op i eindigt of niet. Dus .... voor het slechtste geval is de uiteindelijke $A[i]$ voor $i = 1, \ldots, n - 2$ uniek. Alleen voor de overige twee elementen (dus de twee grootste) hoeft dit niet te gelden. Conclusie: het slechtste geval komt voor d.e.s.d.a. alle elementen van A verschillend zijn, behalve eventueel de grootste waarde. Deze mag twee keer voorkomen.
Het aantal arrayvergelijkingen in de worst case is dan $n - 1 + n - 2 + n - 3 + \cdots + 1 = \frac{1}{2}n(n - 1)$.

c. We bekijken hier het aantal verwisselingen in regel (6) en regel (11). Merk op dat voor elke j hooguit één verwisseling plaatsvindt. Eén als $A[j] < A[i]$, één als $A[j] = A[i]$ en nul als $A[j] > A[i]$. Het minimale aantal verwisselingen zou dus nul kunnen zijn, en dat blijkt inderdaad voor te kunnen komen. We hebben in totaal nul verwisselingen d.e.s.d.a. voor elke i geldt dat voor alle $j = i+1, i+2, \ldots, n$ geldt dat $A[j] > A[i]$. Voor $i = 1$ betekent dat bijvoorbeeld dat: $A[2] > A[1], A[3] > A[1], \ldots, A[n] > A[1]$, met andere woorden: $A[1]$ is echt kleiner dan alle andere array-elementen. Merk ook op dat er in dit geval geen enkele verwisseling plaatsvindt, dus na afloop van de eerste ronde is het array nog in zijn oorspronklijke volgorde. Indien we geen verwisselingen doen is dit het geval na elke ronde. Dus: geen verwisselingen in ronde i ($i = 1, 2, \ldots, n-1$) d.e.s.d.a. $A[i+1] > A[i], A[i+2] > A[i], \ldots, A[n] > A[i]$, met andere woorden d.e.s.d.a. $A[i]$ kleiner is dan alle array-elementen die erachter staan. Als dit voor alle $i = 1, 2, \ldots, n-1$ geldt betekent dit dus dat $A[1] < A[2] < A[3] < \cdots < A[n-1] < A[n]$. We vinden dus: het beste geval is nul verwisselingen, en dat komt voor d.e.s.d.a. het array A oplopend gesorteerd is en geen gelijke waarden bevat, dus strikt stijgend is.

d. Het verwisselen van array-elementen is *geen* goede maat voor de complexiteit. Immers, uit **c** weten we dat voor strikt stijgende rijtjes het algoritme nul verwisselingen doet. Echter, voor zulke invoerrijtjes (alle array-elementen zijn verschillend) doet het algoritme juist $\frac{1}{2}n(n-1)$ arrayvergelijkingen (zie **b**). Dat is een ruim orde-verschil. Het aantal verwisselingen is dus zeker niet maatgevend voor de complexiteit van het algoritme.

Opgave 4. (10+9+2+10)
a. We geven een *niet-deterministisch polynomiaal* algoritme A voor Half-Kliek. A heeft als invoer een ongerichte graaf $\mathcal{G} = (V, E)$ met een even aantal knopen. Er is gegeven dat het aantal knopen $|V|$ bekend is en gelijk is aan n, en ook dat de knopen genummerd zijn met i t/m n. [1]
A doet bijvoorbeeld het volgende:

1. Fase 1 (gokfase)
Er wordt een willekeurige string s gegenereerd.
// Deze s stelt hopelijk een kliek voor in $\mathcal{G}$, bestaande uit $n/2$ knopen
// voor; dit wordt in fase 2 gecontroleerd.

2. Fase 2 (controlefase)
// Hier wordt gecontroleerd of s inderdaad een kliek ter grootte $n/2$ voorstelt.
// s moet in dat geval dus een rijtje van $n/2$ verschillende knopen voorstellen
// die samen een kliek vormen in $\mathcal{G}$. Dit gaan we controleren.

- controleer dat s precies $n/2$ knopen bevat: s aflopen, checken dat elke s_i een waarde tussen 1 en n voorstelt en tellen. Dat kan in $O(|s|)$ stappen.

- controleer dat alle s_i verschillend zijn: s aflopen en voor elke s_i rechts daarvan kijken. Als

[1] Merk op dat het aantal knopen tellen $O(|V|)$ is, dus polynomiaal in $\mathcal{G}$. Het zou dus slechts polynomiaal veel stappen kosten om dit te berekenen. Dit heeft derhalve geen invloed op de polynomialiteit van Fase 2 van het algoritme A. Daarom mochten we wel aannemen dat het aantal knopen bekend is.

je daar dezelfde waarde nog eens tegenkomt bestaat s kennelijk niet uit allemaal verschillende waardes; als dat voor geen der s_i het geval is juist wel. Dat kan in $O(|s|^2)$ stappen.

Als aan deze twee controles is voldaan stelt s een deelverzameling van de knopen voor ter grootte $n/2$.

- controleer dat de knoopverzameling voorgesteld door s een kliek is in $\mathcal{G}$. Er moet dus gecontroleerd worden of er tussen elk tweetal knopen uit s een tak loopt. Dat kan in $O(|s|^2 \cdot |\mathcal{G}|)$ door voor elke s_i in E te kijken of de tak (s_i, s_j) met $j > i$ daar aanwezig is.

Als de drie controles positief zijn stelt s een kliek bestaande uit $|V|/2$ 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. Er geldt dus: Fase 2 geeft True $\iff$ s is kliek ter grootte $|V|/2$ in $\mathcal{G}$.

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

Nu geldt: het antwoord van A op invoer $\mathcal{G}$ is "ja" (per definitie) $\iff$ er is een executie van A die "ja" oplevert $\iff$ er is een string s waarop A in Fase 2 True geeft $\iff$ (ons concrete algoritme) er is een string s die een kliek ter grootte $|V|/2$ in $\mathcal{G}$ voorstelt $\iff$ $\mathcal{G}$ heeft een kliek ter grootte $|V|/2$ $\iff$ $\mathcal{G}$ is een ja-instantie van Half-Kliek.

Kortom, A is inderdaad een (niet-deterministisch) algoritme voor Half-Kliek. Bovendien is het polynomiaal. Immers, voor ja-executies stelt de gegokte s een kliek ter grootte $|V|/2$ voor. In dat geval is dus $|s| = \mathrm{O}(|\mathcal{G}|)$. Die (ja-)executie met die s kan derhalve zeker wel in: $O(|s|)$ (Fase 1) + $O(|s|)$ + $O(|s|^2)$ + $\mathrm{O}(|s|^2 \cdot |\mathcal{G}|)$ (Fase 2) + $\mathrm{O}(1)$ (Fase 3) = $O(|\mathcal{G}|)$ + $O(|\mathcal{G}|^2)$ + $O(|\mathcal{G}|^3)$ = $O(|\mathcal{G}|^3)$ stappen. Er is dus voor ja-instanties een ja-executie die $O(|\mathcal{G}|^3)$ 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 ($O(|\mathcal{G}|^3)$) algoritme voor Half-Kliek.

b.[2]
"$\Longrightarrow$": Stel $\langle \mathcal{G}, k \rangle$ is een ja-instantie van Kliek. Dan bestaat er dus een deelverzameling V' bestaande uit k knopen die een kliek voorstelt in $\mathcal{G}$. Aan te tonen dat $\mathcal{G}'$ dan een kliek heeft, bestaande uit de helft van het aantal knopen (van $\mathcal{G}'$ dus).

- $k \geq \frac{n}{2}$:
  $\mathcal{G}'$ heeft in dit geval $n + 2k - n = 2k$ knopen. We zoeken dus een kliek in $\mathcal{G}'$ met k knopen. Merk op dat V' ook een kliek is in $\mathcal{G}'$; er zijn immers geen knopen of takken verwijderd in de constructie, dus V' is een deelverzameling van de knopen van $\mathcal{G}'$ en er zit in $\mathcal{G}'$ ook nog steeds een tak tussen elk tweetal knopen uit V'.

[2]Speciaal geval: $k = 1$ en $n = 2$. Omdat elke knoop een kliek ter grootte 1 is, is $\mathcal{G}$ altijd een ja-instantie. Verder bevat $\mathcal{G}'$ in dat geval 2 knopen, dus een half-kliek moet uit 1 knoop bestaan en derhalve is ook $\mathcal{G}'$ altijd een ja-instantie.

- $k < \frac{n}{2}$:
  $\mathcal{G}'$ heeft in dit geval $n + n - 2k = 2 \cdot (n - k)$ knopen. We zoeken dus een kliek in $\mathcal{G}'$ met $n - k$ knopen. Merk op dat de knoopverzameling V'' die bestaat uit de k knopen van V' en de $n - 2k$ toegevoegde knopen, precies $k + n - 2k = n - k$ knopen bevat. Tevens zit er tussen elk tweetal knopen uit de oorspronkelijke V' een tak (er zijn immers geen takken verwijderd), maar bovendien is elke toegevoegde knoop met alle andere knopen verbonden. Dus tussen elk tweetal knopen uit V'' zit een tak. Ergo: V'' is een kliek ter grootte $n - k$ in $\mathcal{G}'$.

"$\Longleftarrow$": Stel $\mathcal{G}'$ is een ja-instantie van Half-Kliek. Dan is er dus een deelverzameling V'' van de knopen van $\mathcal{G}'$, bestaande uit de helft van het aantal knopen (van $\mathcal{G}'$ dus) die een kliek voorstelt in $\mathcal{G}'$. Aan te tonen dat $\langle \mathcal{G}, k \rangle$ dan een ja-instantie voor Kliek is, met andere woorden dat $\mathcal{G}$ dan een kliek heeft, bestaande uit k knopen.

- $k \geq \frac{n}{2}$:
  $\mathcal{G}'$ heeft in dit geval $n + 2k - n = 2k$ knopen en heeft een kliek V'' ter grootte k. Merk op dat geen van de toegevoegde (geïsoleerde!) knopen deel uit kan maken van die kliek, want er zit nooit een tak tussen die knoop en welke andere knoop dan ook. Dus, deze kliek bevat alleen knopen van $\mathcal{G}$. Derhalve is V'' ook een kliek ter grootte k in $\mathcal{G}$.

- $k < \frac{n}{2}$:
  $\mathcal{G}'$ heeft in dit geval $n + n - 2k = 2 \cdot (n - k)$ knopen en heeft een kliek V'' ter grootte $n - k$. Merk op dat er slechts $n - 2k$ nieuwe knopen zijn toegevoegd in de constructie van $\mathcal{G}'$. Dat betekent dat V'' altijd ten minste nog k knopen van $\mathcal{G}$ moet bevatten. We kiezen er daar k van en noemen die deelverzameling van k knopen even V'. Het voorgaande houdt in (want V'' is een kliek in $\mathcal{G}'$) dat er in het bijzonder tussen elk tweetal knopen uit V' een tak zit; dit zijn takken die ook al in $\mathcal{G}$ zaten. Conclusie: V' is een kliek ter grootte k in $\mathcal{G}$.

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 Kliek $\in \mathcal{NPC}$. In het bijzonder geldt dus dat Kliek NP-hard is, en derhalve dat alle NP-problemen naar Kliek reduceren. Dat betekent dat ook Half-Kliek $\leq_{\mathrm{P}}$ Kliek, aangezien Half-Kliek $\in \mathcal{NP}$ volgens **a**.
(ii) (1) Uit Half-Kliek $\leq_{\mathrm{P}}$ Kliek volgt -intuïtief- alleen maar dat Half-Kliek hooguit even moeilijk is als Kliek. Kliek is weliswaar heel moeilijk (namelijk $\mathcal{NPC}$), maar dat betekent niet dat Half-Kliek ook heel moeilijk zou moeten zijn. Half-Kliek kan nog best in $\mathcal{P}$ zitten; dat is niet in tegenspraak met **a** en **d(i)**. We kunnen dus niet zomaar concluderen dat Half-Kliek $\in \mathcal{NPC}$.
(2) Uit Kliek $\leq_{\mathrm{P}}$ Half-Kliek (en **a**) volgt dat wel. Immers -intuïtief- Half-Kliek is minstens even moeilijk als Kliek. Kliek is heel moeilijk, dus dan moet Half-Kliek dat ook zijn. Aangezien tevens Half-Kliek $\in \mathcal{NP}$ is de conclusie dat Half-Kliek $\in \mathcal{NPC}$. Een preciezer en beter argument: aangezien Kliek $\in \mathcal{NPC}$ reduceert elk probleem uit $\mathcal{NP}$ polynomiaal naar Kliek. Via de transitiviteit van $\leq_{\mathrm{P}}$ volgt dan uit (#) dat alle NP-problemen ook

polynomiaal naar Half-Kliek reduceren.[3] Conclusie: Half-Kliek is NP-hard. Samen met Half-Kliek $\in \mathcal{NP}$ volgt dat Half-Kliek $\in \mathcal{NPC}$.
(iii) We weten uit het voorgaande dat Half-Kliek $\in \mathcal{NPC}$. Stel nu dat Half-Kliek $\leq_{\mathrm{P}} Q$ voor zekere $Q \in \mathcal{P}$. Dan volgt daar meteen uit dat Half-Kliek $\in \mathcal{P}$, maar ook dat elk NP-probleem in $\mathcal{P}$ zit. Dit omdat vanwege de transitiviteit van $\leq_{\mathrm{P}}$ elk NP-probleem via Half-KLiek naar Q reduceert. Ergo: $\mathcal{NP} \subseteq \mathcal{P}$. Samen met $\mathcal{P} \subseteq \mathcal{NP}$ volgt daar dus uit dat $\mathcal{P} = \mathcal{NP}$.[4] Het is dan ook onwaarschijnlijk dat er zo'n reductie bestaat.
(iv) Merk op dat $Q \in \mathcal{P} \subseteq \mathcal{NP}$. Omdat Half-Kliek NP-hard is (want NP-volledig) reduceert elk probleem in $\mathcal{NP}$ naar Half-Kliek, dus in het bijzonder ook Q. Ergo: er bestaat zeker zo'n polynomiale reductie.

[3]Er mag hier ook direct de volgende stelling uit het college genoemd worden: als P een probleem is waarvoor geldt dat $Q \leq_{\mathrm{P}} P$ voor een of andere $Q \in \mathcal{NPC}$, dan is P NP-hard. Het bewijs van die stelling deden we overigens via de transitiviteit van $\leq_{\mathrm{P}}$.

[4]Of gebruik direct de stelling van college: Als een willekeurig NP-volledig probleem in $\mathcal{P}$ zit, dan is $\mathcal{P} = \mathcal{NP}$.