Tentamen Complexiteit 20 juni 2024
Uitwerking

Opgave 1. (25 punten)

a. (5 punten)

De toernooimethode simuleert als het ware een toernooi tussen de arrayelementen, waarin ze als 'spelers' in rondes een serie 'wedstrijden' (vergelijkingen) spelen waarvan de 'winnaar' (grootste) telkens doorgaat naar de volgende ronde. De uiteindelijke winnaar is het grootste element. Het op-een-na-grootste element is een van de elementen die een wedstrijd hebben verloren van de winnaar. Het wint immers van alle andere elementen.

Concreet: de spelers worden in paren vergeleken, waarna de winnaars doorgaan naar de volgende ronde. In elke ronde spelen de overgebleven elementen weer wedstrijden, totdat er één speler overblijft. In de eerste ronde kost dit $\frac{n}{2}$ vergelijkingen, in de tweede $\frac{n}{4}$, in de derde $\frac{n}{8}$, en zo verder tot één vergelijking in de laatste ronde. In totaal kost dit $n-1$ vergelijkingen. Het aantal spelers wordt elke ronde gehalveerd, dus het aantal rondes is $\lg n$ (merk op dat n hier een tweemacht is, dus we hoeven niet af te ronden). Het op-een-na-grootste element is het grootste van de $\lg n$ elementen die een wedstrijd hebben verloren van de uiteindelijke winnaar; hiervan de grootste vinden kost $\lg n - 1$ vergelijkingen. In totaal dus $n + \lg n - 2$.

Voorbeeld op het rijtje $1, 6, 2, 10, 16, 12, 15, 3, 7, 8, 9, 11, 5, 14, 13, 4$ ($n = 16$):

$$\begin{gathered} 1 < \underline{6} \quad 2 < \underline{10} \quad \underline{16} > 12 \quad \underline{15} > 3 \quad 7 < \underline{8} \quad 9 < \underline{11} \quad 5 < \underline{14} \quad \underline{13} > 4 \\ 6 < \underline{10} \quad \underline{16} > 15 \quad 8 < \underline{11} \quad \underline{14} > 13 \\ 10 < \underline{16} \quad 11 < \underline{14} \\ \underline{16} > 14 \\ 16 \end{gathered}$$

Het grootste element is dus 16, in $8 + 4 + 2 + 1 = 15$ ($= 16 - 1$) vergelijkingen. Het op-een-na-grootste element is het grootste van 12, 15, 10 en 14, de vier elementen die hebben verloren van 16. Dit kost nog eens 3 ($= \lg 16 - 1$) vergelijkingen. Totaal 18 ($= 16 + \lg 16 - 2$) vergelijkingen.

b. (4 punten)

Een pad van de wortel naar een blad stelt de serie achtereenvolgende arrayvergelijkingen voor die het algoritme doet op zekere invoer. De lengte van zo'n pad correspondeert precies met het aantal arrayvergelijkingen dat op die invoer wordt gedaan. In het bijbehorende blad is het antwoord van het algoritme voor die invoer bekend: het algoritme stopt. Bladeren kunnen derhalve geassocieerd worden met de eindantwoorden/uitkomsten die het algoritme vindt. De hoogte van de beslissingsboom (lengte van het langste pad) stelt dan ook het aantal arrayvergelijkingen in de worst case voor.

c. (8 punten)

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 dat er antwoorden mogelijk zijn, dus $b = \#\text{bladeren} \geq$ aantal mogelijke antwoorden.

- Verder hebben we een stelling voor binaire bomen die het verband aangeeft tussen de hoogte h en het aantal bladeren b, namelijk $h \geq \lceil \lg b \rceil$.

We moeten dus voor ons probleem het aantal mogelijke antwoorden weten, in dit geval de indices van de grootste en op-een-na-grootste elementen uit het array.

Omdat we aannemen dat A $\frac{n}{8}$-gesorteerd is, weten we dat het grootste element een van de laatste $\frac{n}{8}$ elementen moet zijn. Elk element op een positie $i \leq n - \frac{n}{8}$ is immers kleiner dan het element op positie $i + \frac{n}{8}$. Er zijn dus $\frac{n}{8}$ mogelijkheden voor het grootste element.

Het op-een-na-grootste element kan dan een van de $\frac{n}{8} - 1$ andere elementen van de laatste $\frac{n}{8}$ zijn, of het element op positie $j - \frac{n}{8}$, waarbij $A[j]$ het grootste element is. Dit is immers wel kleiner dan $A[j]$, maar hoeft niet kleiner te zijn dan de grootste elementen van de andere deelrijtjes. Alle andere elementen (i.h.b. de op-een-na-grootste elementen van de andere deelrijtjes) zijn kleiner dan minstens twee andere elementen (namelijk het grootste van hun deelrijtje en $A[j]$), dus vallen af. Voor elk grootste element zijn er dus ook $\frac{n}{8}$ mogelijkheden voor het op-een-na-grootste element.

Bij elkaar zijn dit dan $\frac{n}{8} \times \frac{n}{8} = \frac{n^2}{64}$ mogelijke antwoorden. Met de eerdere stellingen geeft dit $b \geq \frac{n^2}{64}$ en $h \geq \lceil \lg b \rceil \geq \lceil \lg \frac{n^2}{64} \rceil = \lceil \lg n^2 - \lg 64 \rceil = \lceil 2 \lg n - 6 \rceil = 2 \lg n - 6$. De afrondtekens in de laatste stap kunnen weg omdat n een tweemacht is en $\lg n$ dus geheel is.

d. (3 punten)

De toernooimethode is optimaal voor willekeurige rijtjes, maar de ondergrens uit **c** is voor specifieke rijtjes (namelijk $\frac{n}{8}$-gesorteerd) en maakt dus gebruik van voorkennis. Het zou kunnen dat daarvoor minder vergelijkingen nodig zijn dan voor de toernooimethode, dus alleen op basis daarvan kunnen we niet stellen dat de ondergrens uit **c** niet scherp is.

e. (5 punten)

Zoals uitgelegd bij **c**, komen het grootste en op-een-na-grootste element altijd uit de laatste $\frac{n}{4}$ elementen. Hierop de toernooimethode toepassen geeft dus ook het grootste en op-een-na-grootste element van het hele rijtje, in (zie **a**) $\frac{n}{4} + \lg \frac{n}{4} - 2$ vergelijkingen (met hoogste orde term $\frac{n}{4}$).

Alternatief: het op-een-na-grootste element is ofwel het op-een-na-grootste element van de laatste $\frac{n}{8}$, of het element $\frac{n}{8}$ posities vóór het grootste. Het uitvoeren van de toernooimethode op de laatste $\frac{n}{8}$ elementen geeft het grootste element en het op-een-na-grootste van de laatste $\frac{n}{8}$, in $\frac{n}{8} + \lg \frac{n}{8} - 2$ vergelijkingen. Dan kost het nog één vergelijking om de op-een-na-grootste te vergelijken met het element $\frac{n}{8}$ posities voor het grootste, voor in totaal $\frac{n}{8} + \lg \frac{n}{8} - 1$ vergelijkingen (met hoogste orde term $\frac{n}{8} \leq \frac{n}{4}$).

Opgave 2. (20 punten)

a. (5 punten)

De best case voor het samenvoegen van twee rijtjes van lengtes ℓ en m is $\min(\ell, m)$, wanneer alle elementen van het kortste rijtje kleiner zijn dan alle elementen in het langste rijtje. Dan kosten alle elementen van het kortste rijtje elk één vergelijking om in het hulparray te stoppen, en kunnen alle elementen van het langste rijtje er gratis achter worden geplakt.

Voor dit concrete geval betekent het dat het samenvoegen van I en II $\min(\frac{n}{4}, \frac{n}{4}) = \frac{n}{4}$ vergelijkingen kost in de best case, wanneer alle elementen van één van de twee kleiner zijn dan alle elementen van de ander. (De deelarrays zijn even lang, dus het maakt niet uit van welk van de twee de elementen kleiner zijn. Alle elementen zijn volgens de opgave verschillend, dus we hoeven ook geen rekening te houden met de voorkeur voor links of rechts bij gelijke waarde.) Het samenvoegen

van het resultaat met III kost dan $\min(\frac{2n}{4}, \frac{n}{4}) = \frac{n}{4}$ vergelijkingen in de best case, wanneer alle elementen van III kleiner zijn dan alle elementen in I en II. Ten slotte kost het samenvoegen met IV $\min(\frac{3n}{4}, \frac{n}{4}) = \frac{n}{4}$ vergelijkingen in de best case, wanneer alle elementen van IV kleiner zijn dan alle elementen in I, II en III.

De worst case voor het samenvoegen van twee rijtjes van lengtes ℓ en m is $\ell + m - 1$, wanneer de laatste twee elementen moeten worden vergeleken. Zo worden er zo min mogelijk elementen 'gratis' geplaatst in het hulparray. Dit gebeurt wanneer de laatste elementen van de twee deelarrays ook de twee grootste elementen zijn van het samengevoegde array.

Voor dit concrete geval betekent het dat het samenvoegen van I en II $\frac{n}{4} + \frac{n}{4} - 1$ vergelijkingen kost in de worst case, wanneer $A[\frac{n}{4}]$ en $A[\frac{2n}{4}]$ de grootste elementen zijn uit I en II. Het samenvoegen met III kost dan $\frac{2n}{4} + \frac{n}{4} - 1$ vergelijkingen in de worst case, wanneer $A[\frac{3n}{4}]$ en $\max(A[\frac{n}{4}], A[\frac{2n}{4}])$ de grootste elementen zijn uit I, II en III. Ten slotte kost het samenvoegen met IV $\frac{3n}{4} + \frac{n}{4} - 1$ vergelijkingen in de worst case, wanneer $A[n]$ en $\max(A[\frac{n}{4}], A[\frac{2n}{4}], A[\frac{3n}{4}])$ de grootste elementen zijn uit A. Totaal kost dit dus $\frac{9n}{4} - 3$ vergelijkingen.

b. (10 punten)

We rekenen eerst een paar termen uit:

$$\begin{aligned}
M(n) &= 4M(\frac{n}{4}) + \frac{9}{4}n - 3 && \text{(definitie)}\\
M(\frac{n}{4}) &= 4M(\frac{\frac{n}{4}}{4}) + \frac{9}{4} \times \frac{n}{4} - 3 && \text{(definitie)}\\
&= 4M(\frac{n}{16}) + \frac{9}{16}n - 3 && \text{(herschrijven)}\\
M(\frac{n}{16}) &= 4M(\frac{\frac{n}{16}}{4}) + \frac{9}{4} \times \frac{n}{16} - 3 && \text{(definitie)}\\
&= 4M(\frac{n}{64}) + \frac{9}{64}n - 3 && \text{(herschrijven)}
\end{aligned}$$

Herhaald invullen in de oorspronkelijke betrekking geeft:

$$\begin{aligned}
M(n) &= 4M(\frac{n}{4}) + \frac{9}{4}n - 3 && \text{(definitie)}\\
&= 4[4M(\frac{n}{16}) + \frac{9}{16}n - 3] + \frac{9}{4}n - 3 && \text{(invullen)}\\
&= 16M(\frac{n}{16}) + 4 \times \frac{9}{16}n - 4 \times 3 + \frac{9}{4}n - 3 && \text{(haakjes wegwerken)}\\
&= 16M(\frac{n}{16}) + \frac{9}{4}n - 4 \times 3 + \frac{9}{4}n - 3 && \text{(herschrijven)}\\
&= 16[4M(\frac{n}{64}) + \frac{9}{64}n - 3] + \frac{9}{4}n - 4 \times 3 + \frac{9}{4}n - 3 && \text{(invullen)}\\
&= 64M(\frac{n}{64}) + 16 \times \frac{9}{64}n - 16 \times 3 + \frac{9}{4}n - 4 \times 3 + \frac{9}{4}n - 3 && \text{(haakjes wegwerken)}\\
&= 64M(\frac{n}{64}) + \frac{9}{4}n - 16 \times 3 + \frac{9}{4}n - 4 \times 3 + \frac{9}{4}n - 3 && \text{(herschrijven)}\\
&= 4^3M(\frac{n}{4^3}) + 3 \times \frac{9}{4}n - 3 \times (4^2 + 4^1 + 4^0) && \text{(herschrijven)}
\end{aligned}$$

Dit doet vermoeden dat de algemene vorm als volgt is:

$$\begin{aligned} M(n) &\stackrel{?}{=} 4^\ell M(\frac{n}{4^\ell}) + \ell \times \frac{9}{4}n - 3 \times \sum_{i=0}^{\ell-1} 4^i && \text{(vermoeden)} \\ &= 4^\ell M(\frac{n}{4^\ell}) + \ell \times \frac{9}{4}n - 3 \times \frac{1}{3}(4^\ell - 1) && \text{(gesloten formule sommatie)} \\ &= 4^\ell M(\frac{n}{4^\ell}) + \ell \times \frac{9}{4}n - 4^\ell + 1 && \text{(herschrijven)} \end{aligned}$$

Om van de recurrente term af te komen redeneren we terug naar het basisgeval: $\frac{n}{4^\ell} = 1 \Leftrightarrow n = 4^\ell \Leftrightarrow \ell = k = \log_4 n$ (voor de liefhebbers: $= \frac{\lg n}{\lg 4} = \frac{\lg n}{2}$). Dit invullen ($4^\ell = n$ en $\ell = \log_4 n$) geeft:

$$\begin{aligned} M(n) &\stackrel{?}{=} n \times M(\frac{n}{n}) + \log_4 n \times \frac{9}{4}n - n + 1 && \text{(invullen)} \\ &= \frac{9}{4}n \log_4 n - n + 1 && \text{(herschrijven)} \end{aligned}$$

Rest om te bewijzen met volledige inductie dat $M(n) = \frac{9}{4}n \log_4 n - n + 1$.

Voor het basisgeval $n = 1$ geldt dat $M(1) = 0$ en $\frac{9}{4} \times 1 \times \log_4 1 - 1 + 1 = \frac{9}{4} \times 1 \times 0 - 1 + 1 = 0$. Dit gaat dus op.

Voor het inductieve geval nemen we als inductieaanname: stel dat $M(n) = \frac{9}{4}n \log_4 n - n + 1$ voor alle viermachten $n < N$ en met N ook een viermacht. Dan moeten we laten zien dat $M(N) = \frac{9}{4}N \log_4 N - N + 1$:

$$\begin{aligned} M(N) &= 4M(\frac{N}{4}) + \frac{9}{4}N - 3 && \text{(definitie)} \\ &= 4(\frac{9}{4} \times \frac{N}{4} \log_4 \frac{N}{4} - \frac{N}{4} + 1) + \frac{9}{4}N - 3 && \text{(inductieaanname op } M(\tfrac{N}{4})) \\ &= \frac{9}{4}N \log_4 \frac{N}{4} - N + 4 + \frac{9}{4}N - 3 && \text{(haakjes wegwerken)} \\ &= \frac{9}{4}N(\log_4 N - \log_4 4) - N + \frac{9}{4}N + 1 && \text{(herschrijven)} \\ &= \frac{9}{4}N(\log_4 N - 1) - N + \frac{9}{4}N + 1 && \text{(herschrijven)} \\ &= \frac{9}{4}N \log_4 N - \frac{9}{4}N - N + \frac{9}{4}N + 1 && \text{(haakjes wegwerken)} \\ &= \frac{9}{4}N \log_4 N - N + 1 && \text{(herschrijven)} \end{aligned}$$

Hiermee is bewezen dat $M(n) = \frac{9}{4}n \log_4 n - n + 1$ voor alle $n = 4^k, n \geq 1$.

c. (5 punten)

Een inversie is een paar arrayelementen $(A[i], A[j])$ zodat $i < j$ en $A[i] > A[j]$. Omdat I t/m IV nu elk aflopend zijn gesorteerd, hebben we sowieso een inversie voor elk paar elementen binnen elk van I t/m IV. Dit zijn er dan $\frac{1}{2} \times \frac{n}{4}(\frac{n}{4} - 1)$ per deelarray (het aantal geordende paren $(A[i], A[j])$ uit $\frac{n}{4}$ elementen waarvoor $i < j$). In totaal dus minstens $4 \times \frac{1}{2} \times \frac{n}{4}(\frac{n}{4} - 1) = \frac{1}{2}n(\frac{n}{4} - 1)$. Voor het minimale aantal inversies kunnen we aannemen dat er geen inversies zijn tussen elementen van

verschillende deelarrays, wat het geval is als I, II, III en IV onderling oplopend zijn gesorteerd (alle elementen uit I kleiner dan alle elementen uit II enzovoorts).

Insertion sort vergelijkt en verwisselt alleen aangrenzende elementen en kan zodoende hooguit één inversie oplossen per vergelijking. Het zal dus ook altijd minstens $\frac{1}{2}n(\frac{n}{4}-1)$ vergelijkingen moeten doen om A te sorteren.

Opgave 3. (25 punten)

a. (5 punten)

We geven per i (1 of 2) een tabel met daarin de iteraties van de binnenste loop en de waardes na elke iteratie. Veranderingen in $B[i]$, *links* en *rechts* zijn onderstreept. Merk op dat de waardes in de laatste iteratie niet veranderen omdat de loop hier stopt. Dit telt echter wel als vergelijking. Merk ook op dat, bij $i = 2$, *links* eindigt op positie 9 ($= n + 1$).

Voor de volledigheid het rijtje: 7, 1, 3, 3, 10, 1, 2, 2.

Voor $i = 1$, $A[1] = 7$.

Vergelijkingen	$B[i]$	*links*	*rechts*	*som*
	0	2	2	1
1	0	2	<u>3</u>	4
2	0	2	<u>4</u>	7
3	<u>1</u>	<u>3</u>	4	6
4	1	3	<u>5</u>	16
5	1	<u>4</u>	5	13
6	1	<u>5</u>	5	10
7	1	<u>6</u>	5	0
8	1	6	<u>6</u>	1
9	1	6	<u>7</u>	3
10	1	6	<u>8</u>	5
11	1	6	8	5

Voor $i = 2$, $A[2] = 1$.

Vergelijkingen	$B[i]$	*links*	*rechts*	*som*
	0	3	3	3
1	0	<u>4</u>	3	0
2	0	4	<u>4</u>	3
3	0	<u>5</u>	4	0
4	0	5	<u>5</u>	10
5	0	<u>6</u>	5	0
6	0	5	<u>6</u>	1
7	<u>1</u>	<u>7</u>	6	0
8	1	7	<u>7</u>	2
9	1	<u>8</u>	7	0
10	1	8	<u>8</u>	2
11	1	<u>9</u>	8	0
12	1	9	8	0

b. (10 punten)

Merk op dat de buitenste loop (regel 2) in feite een for-loop is voor $i = 1, \ldots, n-1$.

In de best case wordt de arrayvergelijking op regel 4 zo min mogelijk gedaan. Hiervoor moet voor elke i zo snel mogelijk worden bereikt dat de twee condities *beide* onwaar zijn zodat de binnenste loop termineert (beide omdat het een *or* is). Met andere woorden: we willen zo snel mogelijk dat, tegelijkertijd, $som < A[i]$ én $rechts \geq n$. Hoe dan ook zal *rechts* moeten worden opgehoogd van $i+1$ naar n, anders termineert de loop niet. (Merk op dat *rechts* nooit groter kan worden dan n: *rechts* wordt alleen opgehoogd, met 1, als $som < A[i]$, in welk geval $rechts < n$ moet gelden om de loop te betreden.)

rechts wordt opgehoogd wanneer $som < A[i]$. Elke keer dat dit onwaar is, wordt *links* opgehoogd in plaats van *rechts*, wat een extra vergelijking kost. In het beste geval wordt *links* dus helemaal nooit opgehoogd, wat het geval is wanneer *som* altijd kleiner blijft dan $A[i]$. Dan termineert de binnenste loop ook zodra $rechts = n$, d.w.z., na $n-i$ vergelijkingen ($n-i-1$ waarin *rechts* wordt opgehoogd, en 1 extra om te stoppen, zoals opgemerkt bij **a**). Dit gebeurt wanneer $A[i] > A[i+1] + \ldots + A[n]$ voor alle $i < n$.

In totaal kost dit dan $(n-1) + (n-2) + (n-3) + \ldots + 1 = \sum_{i=1}^{n-1} i = \frac{1}{2}n(n-1)$ vergelijkingen.

c. (10 punten)

In de worst case wordt de arrayvergelijking op regel 4 zo vaak mogelijk gedaan. Hiervoor moet zo lang mogelijk worden uitgesteld dat beide condities onwaar worden. Met andere woorden: we willen zo lang mogelijk dat $som \geq A[i]$ óf dat $rechts < n$. Uiteindelijk wordt *rechts* sowieso opgehoogd tot n. In de iteraties waarin *rechts* niet wordt opgehoogd, wordt *links* opgehoogd; dit willen we zo vaak mogelijk doen. Merk op dat het niet uitmaakt *wanneer links* wordt opgehoogd (zoals om en om met *rechts*), zolang het uiteindelijk maar zo vaak mogelijk gebeurt.

We willen dus zo vaak mogelijk dat $som = A[links] + \ldots + A[rechts] \geq A[i]$ zodat *links* wordt opgehoogd. In het bijzonder moet dit ook zo lang mogelijk blijven gelden wanneer $rechts = n$ (wat vroeg of laat gebeurt), d.w.z., $A[links] + \ldots + A[n] \geq A[i]$. Zoals gezien bij **a**, kan dit zelfs nog gelden wanneer $links = n$, wanneer $A[n] \geq A[i]$; *links* wordt dan opgehoogd tot $n+1$. Dit dekt meteen alle gevallen: als $A[n] \geq A[i]$, moet *links* altijd worden opgehoogd tot $n+1$. Merk op dat *links* niet maximaal wordt opgehoogd wanneer $A[n] < A[i]$. Merk op dat *links* nooit groter kan worden dan $n+1$: wanneer $links = n+1$ (en $rechts = n$) is *som* gelijk aan 0, waardoor de loop termineert.

Alles bij elkaar kost dit $2(n-i)$ vergelijkingen per i: $n-i-1$ voor het ophogen van *rechts* van $i+1$ naar n; $n-i$ voor het ophogen van *links* van $i+1$ naar $n+1$; 1 om te stoppen. Dit zijn precies twee keer zo veel vergelijkingen als bij **b**, dus $n(n-1)$.

Opgave 4. (30 punten)

a. (8 punten)

We geven een *niet-deterministisch polynomiaal* algoritme A voor 1-in-3-SAT. A heeft als invoer een logische formule ϕ in 3-CNF. We mogen aannemen (zie de opgave) dat het aantal verschillende logische variabelen in ϕ bekend is en gelijk is aan n.

A doet bijvoorbeeld het volgende:

1. Fase 1 (gokfase)

   Er wordt een string s gegenereerd, hierna te interpreteren als een rij booleans (True, False) $s_1, \ldots, s_m$. Deze zou een waardering moeten voorstellen van de logische variabelen in ϕ die van elke clausule precies één literal waarmaakt; dit wordt gecontroleerd in fase 2. Het genereren van s kan in $O(|s|)$.

2. Fase 2 (verificatiefase)

   Er wordt gecontroleerd of s inderdaad een waardering voorstelt van de logische variabelen in ϕ die van elke clausule precies één literaal waarmaakt. De string s moet in dat geval dus een rijtje van n booleans voorstellen. Concreet voeren we de volgende controles uit:

   (a) we controleren dat s precies n waardes bevat: s aflopen en tellen. Dit kan in $O(|s|)$ stappen.

   (b) we controleren dat alle s_i booleans voorstellen (True, False) en dus een waardering van een logische variabele: s aflopen en controleren of voor elke s_i geldt dat s_i True of False is. Dit kan in $O(|s|)$ stappen.

   Als aan deze voorwaarden is voldaan stelt s een waardering voor van de logische variabelen in ϕ. Dit is dus een soort syntactische controle. Dan resteert de belangrijkste controle, namelijk dat s van elke clausule in ϕ precies één literal waarmaakt:

   (c) we controleren dat van elke clausule in ϕ precies één literal wordt waargemaakt onder de waardering (voorgesteld door) s: ϕ aflopen en voor elke clausule de waardering van de logische variabelen van de drie literals opzoeken, en controleren of (na het verwerken van negaties) er precies één literal waar is. Dit kan in $O(|\phi| \times |s|)$ stappen.

   Als de drie controles positief zijn, stelt s een waardering voor van de logische variabelen in ϕ die van elke clausule precies één literal waarmaakt, en retourneert fase 2 True. Zodra een van de controles niet klopt, wordt False geretourneerd of beginnen we een oneindige loop.

3. Fase 3 (uitvoerfase)

   Als fase 2 True oplevert, wordt "ja" uitgevoerd. Anders is er geen uitvoer.

Nu geldt: fase 2 geeft True $\Leftrightarrow$ s stelt een waardering voor van de logische variabelen in ϕ die van elke clausule precies één literal waarmaakt. En dus:

het antwoord van A op invoer ϕ is "ja"
$\Leftrightarrow$ er is een executie van A die "ja" oplevert (definitie)
$\Leftrightarrow$ er is een string s waarop A in fase 2 True geeft
$\Leftrightarrow$ er is een string s die een waardering voorstelt van de logische variabelen in ϕ
die van elke clausule precies één literal waarmaakt (ons concrete algoritme)
$\Leftrightarrow$ er bestaat een waardering van de logische variabelen in ϕ
die van elke clausule precies één literal waarmaakt
$\Leftrightarrow$ ϕ is een ja-instantie van 1-in-3-SAT

Kortom, A is inderdaad een (niet-deterministisch) algoritme voor 1-in-3-SAT. Bovendien is het polynomiaal. Voor ja-executies stelt de gegokte s een waardering van de logische variabelen in ϕ voor. In dat geval is dus $|s| \in O(|\phi|)$. De (ja-)executie met die s kan dus zeker wel in: $O(|s|)$ (fase 1) $+ O(|s| + O(|s| + O(|\phi| \times |s|)$ (fase 2) $+ O(1)$ (fase 3) $\subseteq O(|\phi|) + O(|\phi|) + O(|\phi|) + O(|\phi| \times |\phi|) + O(1) \subseteq O(|\phi|^2)$ stappen. Er is dus voor ja-instanties een ja-executie die een aantal stappen doet dat polynomiaal is in $|\phi|$, de grootte van de invoer. Met andere woorden: A is polynomiaal.

Conclusie: A is een niet-deterministisch polynomiaal ($O(|\phi|^2)$) algoritme voor 1-in-3-SAT, dus 1-in-3-SAT $\in \mathcal{NP}$.

b. (5 punten)

Als ϕ een ja-instantie is van 3SAT, dan bestaat er een waardering w van de in ϕ voorkomende logische variabelen die ϕ waarmaakt, d.w.z., die van elke clausule in ϕ minstens één literal waarmaakt. Hieruit construeren we een waardering van de in $T(\phi)$ voorkomende logische variabelen die van elke clausule *precies* één literal waarmaakt.

Zoals in de hint: we bekijken één clausule, $C = (\ell_1 \vee \ell_2 \vee \ell_3)$ en onderscheiden de mogelijke gevallen voor de waardering van ℓ_1, ℓ_2 en ℓ_3. We weten dat minstens één van de drie waar moet zijn, dus we kunnen het geval negeren waarin ze alle drie onwaar zijn. In alle gevallen geven we de variabelen die horen bij ℓ_1, ℓ_2 en ℓ_3 dezelfde waarde als in w.

Voor de volledigheid: $T(C) = (\neg\ell_1 \vee a \vee b) \wedge (b \vee \ell_2 \vee c) \wedge (c \vee d \vee \neg\ell_3)$.

- Als $\ell_1 = \text{True}$ en $\ell_2 = \ell_3 = \text{False}$, nemen we $b = \text{True}$ en $a = c = d = \text{False}$. Dit maakt in de eerste clausule alleen b waar, in de tweede clausule ook alleen b, en in de derde clausule alleen $\neg\ell_3$.
- Als $\ell_1 = \ell_2 = \text{True}$ en $\ell_3 = \text{False}$, nemen we $a = \text{True}$ en $b = c = d = \text{False}$. Dit maakt in de eerste clausule alleen a waar, in de tweede clausule alleen ℓ_2 en in de derde clausule alleen $\neg\ell_3$.
- Als $\ell_1 = \ell_3 = \text{True}$ en $\ell_2 = \text{False}$, nemen we (bijvoorbeeld) $a = c = \text{True}$ en $b = d = \text{False}$. Dit maakt in de eerste clausule alleen a waar, in de tweede clausule alleen c, en in de derde clausule alleen c.
- Als $\ell_1 = \ell_2 = \ell_3 = \text{True}$, nemen we $a = d = \text{True}$ en $b = c = \text{False}$. Dit maakt in de eerste clausule alleen a waar, in de tweede clausule alleen ℓ_2, en in de derde clausule alleen d.
- Als $\ell_2 = \text{True}$ en $\ell_1 = \ell_3 = \text{False}$, nemen we $a = b = c = d = \text{False}$. Dit maakt in de eerste clausule alleen $\neg\ell_1$ waar, in de tweede clausule alleen ℓ_2, en in de derde clausule alleen $\neg\ell_3$.
- Het geval $\ell_2 = \ell_3 = \text{True}$ en $\ell_1 = \text{False}$ is analoog aan dat waar $\ell_1 = \ell_2 = \text{True}$ en $\ell_3 = \text{False}$.
- Het geval $\ell_3 = \text{True}$ en $\ell_1 = \ell_2 = \text{False}$ is analoog aan dat waar $\ell_1 = \text{True}$ en $\ell_2 = \ell_3 = \text{False}$.

Merk op dat ℓ_1, ℓ_2 en ℓ_3 hierin dezelfde waarde hebben als die in w, dus dit is consistent tussen de verschillende clausules uit ϕ. De variabelen a, b, c en d zijn fris en komen dus niet voor in (de afbeelding van) andere clausules. De zo geconstrueerde waardering is dus consistent en maakt van elke clausule in $T(\phi)$ precies één literal waar. $T(\phi)$ is dus een ja-instantie van 1-in-3-SAT.

c. (5 punten)

Als $T(\phi)$ een ja-instantie is van 1-in-3-SAT, dan bestaat er een waardering w van de in $T(\phi)$ voorkomende logische variabelen die van elke clausule in $T(\phi)$ precies één literal waarmaakt. Hieruit construeren we een waarmakende waardering voor ϕ: we geven alle logische variabelen in ϕ dezelfde waarde die ze ook hebben in w.

We bekijken de afbeelding van één clausule: $T(C) = (\neg\ell_1 \vee a \vee b) \wedge (b \vee \ell_2 \vee c) \wedge (c \vee d \vee \neg\ell_3)$.

We weten dat w van elk van deze clausules precies één literal waarmaakt. Uit het ongerijmde: stel dat, onder de waardering w, $\ell_1 = \ell_2 = \ell_3 = \text{False}$. Dan volgt dat $a = b = c = d = \text{False}$ om te voldoen aan de eerste en derde clausules. Daarmee wordt echter niet voldaan aan de tweede clausule. Er volgt dus dat w minstens één van ℓ_1, ℓ_2 en ℓ_3 moet waarmaken. De waardering maakt dus ook de oorspronkelijke clausule C in ϕ waar.

Alternatief (niet uit het ongerijmde): als (onder w) $\ell_1 = \text{True}$ of $\ell_3 = \text{True}$, dan zijn we meteen klaar ($\ell_1 \vee \ell_2 \vee \ell_3$ is dan ook waar). Zo niet, dan volgt $a = b = c = d = \text{False}$ vanwege de eerste en derde clausules, en dan volgt uit de tweede clausule dat $\ell_2 = \text{True}$. Er is dus altijd minstens één van ℓ_1, ℓ_2 en ℓ_3 waar.

Merk op dat de zo geconstrueerde waardering alle variabelen in ϕ dezelfde waarde geeft als in w. Per aanname is w consistent, dus de geconstrueerde waardering ook.

d. (2 punten)

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

e. (10 punten)

(i) We weten uit (1) dat 3SAT $\in \mathcal{NPC}$, dus in het bijzonder is 3SAT NP-hard (**d**), wat betekent (**d**) elk probleem in $\mathcal{NP}$ er polynomiaal naar is te reduceren. In het bijzonder geldt dit voor 1-in-3-SAT, dat volgens (2) in $\mathcal{NP}$ zit.

(ii) Dit volgt uit (1), (2) en (3): om NP-volledig te zijn, moet 1-in-3-SAT NP-hard zijn en in $\mathcal{NP}$ zitten. (2) stelt dat 1-in-3-SAT $\in \mathcal{NP}$. Verder volgt uit (1), zoals uitgelegd bij (i), dat elk probleem in $\mathcal{NP}$ polynomiaal is te reduceren naar 3SAT. (3) stelt dat 3SAT polynomiaal is te reduceren naar 1-in-3-SAT. Per transitiviteit van polynomiale reducties is dan elk probleem in $\mathcal{NP}$ polynomiaal te reduceren naar 1-in-3-SAT, dus 1-in-3-SAT is NP-hard. Dus ook NP-volledig.

(iii) Het bepalen of er een waarmakende waardering bestaat voor een formule in 3CNF is een NP-volledig probleem, maar hetzelfde probleem is (deterministisch) polynomiaal op te lossen voor een formule in DNF. Een polynomiale transformatie van een formule in 3CNF naar een (gelijkwaardige) formule in DNF zou dus bewijzen dat $\mathcal{P} = \mathcal{NP}$ (volgens een stelling op college/uit het dictaat, omdat je dan elk probleem in $\mathcal{NP}$ polynomiaal zou kunnen reduceren naar een probleem in $\mathcal{P}$). Dit is mogelijk maar hoogstwaarschijnlijk onwaar.