Complexiteit 2025
Tentamen

Uitwerkingen

Opgave 1
Gegeven een willekeurig array met n verschillende waarden. Dan zijn er $n!$ ordeningen mogelijk, die bij aanvang allemaal voor kunnen komen. Elk sorteeralgoritme moet voor elke invoer het juiste antwoord geven, dus in dit geval de juiste ordening vinden. Bij aanvang zijn dus $n!$ antwoorden mogelijk. De adversary wil het algoritme zo veel mogelijk vragen (hier: arrayvergelijkingen) laten stellen en heeft daarvoor de volgende strategie.

Als het algoritme vraagt of $A[i] < A[j]$, dan kijkt de adversary voor hoeveel mogelijke ordeningen het antwoord "ja" is en voor hoeveel "nee". Hij geeft vervolgens het antwoord dat zo veel mogelijk mogelijke antwoorden (ordeningen) overlaat (en bij evenveel zegt hij bijvoorbeeld altijd "ja"). De ordeningen die corresponderen met het andere antwoord worden weggegooid. Zo zal hij altijd consistent met eerder gegeven antwoorden antwoorden.

De adversary houdt op deze manier na elke stap (arrayvergelijking door het algoritme) altijd ten minste de helft van de mogelijkheden over en bouwt feitelijk een bad case invoer tegen het algoritme op, waarop dat algoritme lang werk heeft.
Het algoritme heeft het antwoord gevonden als er nog maar 1 mogelijke ordening over is, en dat gebeurt tegen deze strategie pas na ten minste $\lceil \lg n! \rceil$ vragen, omdat na elke vergelijking altijd minstens de helft van de mogelijkheden overblijft.
Aangezien deze strategie tegen *elk* algoritme werkt, heeft *elk* algoritme in de worst case ten minste $\lceil \lg n! \rceil$ arrayvergelijkingen nodig (immers voor de opgebouwde bad case kostte het al minstens zoveel vergelijkingen).

Opgave 2
a. 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.

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 dat er antwoorden mogelijk zijn, dus $b = \#\text{bladeren} \geqslant$ 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 $h \geqslant \lceil \lg b \rceil$.

We bepalen het aantal mogelijke ordeningen (volgordes) voor de gegeven soort invoerrijtjes. Tussen de verschillende deelrijtjes ligt de ordening vast, maar binnen elk deelrijtje is elke ordening nog mogelijk. Per deelrijtje zijn er $3! = 6$ mogelijke volgordes. Dit betekent dat het aantal mogelijke ordeningen voor dit soort rijtjes $6 \cdot 6 \cdot 6 \cdots 6 = 6^{\frac{n}{3}}$ is. (Immers, bij elke ordening van het eerste deelrijtje kun je 6 verschillende mogelijke ordeningen

van het tweede rijtje hebben, terwijl de volgorde *tussen* beide rijtjes vastligt. Dat geeft $6 \cdot 6$ mogelijke volgordes voor eerste zes arrayelementen samen. Etcetera.) Het gevolg van dit alles is: #arrayvergelijkingen in de worst case $= h \geqslant \lceil \lg b \rceil \geqslant \lceil \lg 6^{\frac{n}{3}} \rceil = \lceil \frac{n}{3} \lg 6 \rceil = \lceil \frac{n}{3}(\lg 2 + \lg 3) \rceil = \lceil \frac{n}{3}(1 + \lg 3) \rceil (\geqslant \frac{n}{3}(1 + \lg 3))$.

Een foute redenering is om uit te gaan van een beslissingsboom voor $n = 3$, waaruit een ondergrens op het aantal arrayvergelijkingen in de worst case van $\lceil (1 + \lg 3) \rceil$ komt, en vervolgens te zeggen dat er $\frac{n}{3}$ rijtjes van 3 elementen zijn, en dus ten minste $\frac{n}{3} \cdot \lceil (1+\lg 3) \rceil$ arrayvergelijkingen voor onze rijtjes. Dit is verkeerd, aangezien je dan uitgaat van een bepaalde sorteermethode, namelijk het sorteren van elk rijtje van 3 afzonderlijk. De ondergrens moet echter gelden voor elk algoritme.

c. Er kan een algoritme bestaan dat in de worst case $\Theta(n^2)$ arrayvergelijkingen doet, aangezien $\Theta(n^2)$ boven de de ondergrens uit **b** ligt. (Ander argument: ja, dat kan, want er bestaan $\Theta(n^2)$-sorteeralgoritmen (zie college). Daar moet je overigens wel mee oppassen, want het zou best kunnen zijn dat een algoritme dat voor algemene invoer werkt en dat in $\Theta(n^2)$ zit, voor onze speciale rijtjes echt minder dan $\Theta(n^2)$ arrayvergelijkingen doet in de worst case. Zie bijvoorbeeld vraag **d**.)
Dat $\Theta(n^2)$-algoritme zou optimaal kunnen zijn, immers de ondergrens uit **b** hoeft niet scherp te zijn, d.w.z. er hoeft geen algoritme te zijn die $\lceil \frac{n}{3}(1 + \lg 3) \rceil$ haalt.
Ander antwoord: nee, niet optimaal. Immers bijvoorbeeld Mergesort doet $\Theta(n \lg n)$ arrayvergelijkingen op algemene rijtjes, en dus$O(n \lg n)$ op onze rijtjes. Het kan dus zeker beter dan $\Theta(n^2)$.

d. Quicksort: op het oplopend gesorteerde rijtje doet Quicksort $\Theta(n^2)$ arrayvergelijkingen (zie college). Dat rijtje valt onder de speciale rijtjes uit deze opgave. Dat betekent dat Quicksort dus nog steeds in de worst case $\Theta(n^2)$ arrayvergelijkingen zal doen voor de gegeven soort invoerrijtjes.

Insertion sort: deze sorteermethode voegt een voor een de arrayelementen in in het reeds gesorteerde voorstuk. Vanwege de speciale vorm van het array zal elk array-element maar maximaal 3 vergelijkingen nodig hebben om op de juiste plaats terecht te komen (probeer maar). Dat betekent dat Insertion sort in totaal maximaal $3n$ arrayvergelijkingen doet, dus dat het worst case aantal arrayvergelijkingen in $O(n)$ zit. Dit is dus echt beter dan $\Theta(n)$.

Opgave 3
Merk op dat de buitenste while eigenlijk een for-loop is, dus de body daarvan wordt voor elke i 1 keer gedaan.

a. Aangezien $t \geqslant n \geqslant 1 > 0$ is *hoeveel* $< t$ direct na regel (3). Bovendien geldt dan dat $j = i \leqslant n$. Dat betekent dat voor elke i geldt dat de test in regel (4) altijd de eerste keer True is. Derhalve wordt regel (5) voor elke i ten minste één keer uitgevoerd en dus in totaal ten minste n keer.

We kijken nu hoe vaak elke regel plaatsvindt en vergelijken dat met het aantal keer dat regel (5) gebeurt. We moeten laten zien dat alle andere regels / operaties hooguit even vaak plaatsvinden als regel (5) in orde van grootte.

$\#(1) = 1 \leqslant n \leqslant \#(5)$
$\#(2) = n + 1 \leqslant 2n \leqslant 2 \cdot \#(5)$[1]
$\#(3) = n \leqslant \#(5)$
$\#(6) = \#(5)$
$\#(7) = \#(3) \leqslant \#(5)$

[1] Als je hier direct zegt dat $n + 1 \in \Theta(n)$ en dus in orde van grootte wel kleiner of gelijk aan $\#(5)$ wordt dit ook wel goedgerekend.

$\#(8) \leqslant \#(7) \leqslant \#(5)$
$\#(9) = \#(7) \leqslant \#(5)$

De enige regel waar we echt iets mee moeten is regel (4). Immers deze vindt per i altijd precies 1 keer vaker plaats dan regel (5), dus in totaal is $\#(4) = \#(5) + n$. Maar: $\#(4) = \#(5) + n \leqslant \#(5) + \#(5) = 2 \cdot \#(5)$, dus in orde van grootte vindt regel (4) hooguit even vaak plaats als regel (5).

b. We zagen al dat $\#(5) \geqslant n$. Dit aantal (n optellingen) kan ook gehaald worden en dit is het geval $\iff$ voor elke i wordt de eerste ronde van de binnenste while loop (als $j = i$) uitgevoerd en daarna is meteen $hoeveel \geqslant t$. Op dat moment is $hoeveel = A[i]$, dus er moet gelden dat $A[i] \geqslant t$ voor alle i. Echter merk op dat voor $i = n$ de while-loop van regel (4) stopt omdat $j = i + 1 > n$ is geworden, waardoor de tweede test niet meer gebeurt. Dus het maakt niet uit wat de precieze waarde van $A[i]$ is, je doet voor $i = n$ altijd slechts 1 optelling. Dus in de best case n optellingen en dat komt voor d.e.s.d.a. $A[i] \geqslant t$ voor $i = 1, \ldots, n-1$.

c. Het slechtste geval komt voor d.e.s.d.a. voor elke $i = 1, \ldots, n$ de body van de binnenste while zo vaak mogelijk, dus tot en met $j = n$ gedaan wordt, dus dat $hoeveel < t$ in regel (4) voor elke $j = i, \ldots, n$. Merk nu op dat j wordt opgehoogd direct nadat $hoeveel$ is opgehoogd met (de oude) $A[j]$. Dus op het moment van de test in regel (4) is $hoeveel = A[i] + \cdots + A[j-1]$. Dat betekent dat voor het slechtse geval moet gelden dat $A[i] + \cdots + A[j-1] < t$ voor elke $j = i, \ldots, n$, en dat voor elke i. Omdat alle $A[j]$ bovendien > 0 zijn betekent dit: worst case komt voor d.e.s.d.a. $A[i] + \cdots + A[n-1] < t$ voor alle i d.e.s.d.a. $A[1] + \cdots + A[n-1] < t$. We zien weer dat de waarde van $A[n]$ niet uitmaakt; daarvoor doe je altijd 1 optelling, ongeacht de waarde van $A[n]$.

Het aantal optellingen dat gebeurt in de worst case (dus voor dit soort rijtjes) is $n + n - 1 + n - 2 + \cdots + 2 + 1 = \frac{1}{2}n(n+1)$.

Opgave 4

a. Als er slechts één arrayelement is, geven we dit terug, wat in 1 stap geschiedt.

Stel nu dat $n > 1$. Het array wordt dan in twee even grote delen gesplitst, met $\frac{n}{2}$ elementen elk. Op beide helften wordt het algoritme recursief aangeroepen, om de beste rijtjes volledig bevat in de linker- en rechterhelft te vinden. Dit geeft dus $2T(\frac{n}{2})$ stappen.

Om het grootste rijtje te vinden dat het midden van het array passeert, zoeken we eerst het grootste rijtje dat vanaf het midden naar links loopt. Hiertoe vergelijken we $\frac{n}{2} + 1$ rijtjes (namelijk inclusief het lege rijtje), wat $\frac{n}{2}$ stappen kost. Vervolgens doen we hetzelfde aan de rechterkant: weer $\frac{n}{2}$ stappen. Dan tellen we de twee grootste halve rijtjes bij elkaar op: nog één stap. In totaal dus $\frac{n}{2} + \frac{n}{2} + 1 = n + 1$ stappen.

Ten slotte bepalen we het maximum van de drie kandidaatrijtjes, wat nog eens 2 stappen kost. Bij elkaar opgeteld komen we tot de gewenste recurrente betrekking.

b. Zij $n = 2^k$. We vullen de betrekking herhaaldelijk in zichzelf in:

$$\begin{aligned}
T(n) &= 2T(\tfrac{n}{2}) + n + 3 \\
&= 2\left(2T(\tfrac{n}{4}) + \tfrac{n}{2} + 3\right) + n + 3 \\
&= 2^2 T(\tfrac{n}{4}) + 2n + 6 + 3 \\
&= 2^2\left(2T(\tfrac{n}{8}) + \tfrac{n}{4} + 3\right) + 2n + 6 + 3 \\
&= 2^3 T(\tfrac{n}{8}) + 3n + 12 + 6 + 3 \\
&= 2^3 T(\tfrac{n}{8}) + 3n + 3(4 + 2 + 1) \\
&= \ldots \\
&= 2^\ell T(\tfrac{n}{2^\ell}) + \ell n + 3\sum_{i=1}^{\ell-1} 2^i \\
&= 2^\ell T(\tfrac{n}{2^\ell}) + \ell n + 3(2^\ell - 1).
\end{aligned}$$

Om terug te komen naar het basisgeval $T(1)$, kiezen we $2^\ell = n$, oftewel $\ell = \lg n$. Dit resulteert in

$$T(n) = 2^{\lg n} T(1) + n \lg n + 3(2^{\lg n} - 1) = n + n \lg n + 3(n-1) = 4n + n \lg n - 3.$$

We bewijzen ons vermoeden met volledige inductie. Voor $n = 1$ is de formule juist:

$$4 \cdot 1 + 1 \cdot \lg 1 - 3 = 4 + 1 \cdot 0 - 3 = 1 = T(1).$$

Laat nu $N > 1$ een tweemacht, en neem aan dat de formule juist is voor alle tweemachten $n < N$. We berekenen

$$\begin{aligned}
T(N) &= 2T(\tfrac{N}{2}) + N + 3 \\
&= 2\left(4\tfrac{N}{2} + \tfrac{N}{2}\lg\tfrac{N}{2} - 3\right) + N + 3 \\
&= 4N + N(\lg N - \lg 2) - 6 + N + 3 \\
&= 4N + N\lg N - N - 3 + N \\
&= 4N + N\lg N - 3.
\end{aligned}$$

Hiermee is het bewijs voltooid.

Opgave 5

a. We geven een algoritme in drie fases.

Fase 1: gokfase. We schrijven een willekeurige string s, verder te interpreteren als rij gehele getallen.

Fase 2: verificatiefase. We verifiëren of s een oplossing van de invoerinstantie representeert, d.w.z. een deelverzameling van de knopen die voldoet aan de gestelde eisen. Hiertoe controleren we de volgende zaken.

1. Staan er precies n getallen? Hiervoor lopen we hoogstens de hele string door, dus $O(|s|)$.
2. Is elk getal minstens 1 en hoogstens n? Wederom $O(|s|)$.
3. Komt ieder getal tussen 1 en n precies één keer voor? Een naïeve implementatie vergelijkt elk paar getallen in s, wat $O(|s|^2)$ stappen geeft; het kan sneller, maar in ieder geval polynomiaal in $|s|$.

4. Geeft de verzameling knopen gerepresenteerd in s een dominating set voor $\mathcal{G}$? Hiertoe moeten we controleren of iedere knoop in V ofwel bevat is in s, ofwel verbonden is met een knoop in s. Het eerste kunnen we controleren door voor iedere knoop in V het rijtje s langs te lopen in $O(|V| \cdot |s|)$ stappen. Het tweede kunnen we controleren door voor iedere knoop in V alle takken te bekijken, en voor elke tak het andere uiteinde in s te zoeken: $O(|V| \cdot |E| \cdot |s|)$ stappen. Andere implementaties die een polynomiaal aantal stappen kosten, zijn ook mogelijk.

Als elk van de vier tests doorstaan wordt, geven we "true" als uitvoer. Anders "false" of beginnen we een oneindige loop.

Fase 3: uitvoerfase. Als er in fase 2 "true" werd geretourneerd, antwoorden we "ja". Anders geen uitvoer.

Er geldt nu dat in de derde fase "ja" wordt geantwoord d.e.s.d.a. de gegenereerde string s een dominating set voor $\mathcal{G}$ voorstelt, wat mogelijk is d.e.s.d.a. de invoer $(\mathcal{G}, k)$ een ja-instantie is voor het probleem.

b. (5 pt.) Stel dat ϕ een ja-instantie is voor 3-SAT, en laat $e(x_i) \in \{\text{true}, \text{false}\}$ een waarheidstoekenning van de x_i zijn die ϕ waar maakt. Construeer nu de verzameling

$$U = \{v_i^t \mid e(x_i) = \text{true}\} \cup \{v_i^f \mid e(x_i) = \text{false}.$$

In woorden: we kiezen uit ieder drietal $\{v_i^t, v_i^f, v_i^0\}$ voor onze dominating set U de knoop v_i^t als x_i op true is gezet, of v_i^f als x_i op false is gezet. We gaan na dat U dan inderdaad een dominating set van $T(\phi)$ is.

Merk allereerst op dat U per constructie exact één knoop uit elk drietal knopen $\{v_i^t, v_i^f, v_i^0\}$ bevat. Er zijn precies n van deze drietallen, dus $|U| = n$ voldoet qua grootte.

Gezien de drietallen $\{v_i^t, v_i^f, v_i^0\}$ elk onderling verbonden zijn, is elk van deze knopen ofwel bevat in U, ofwel een buur van een knoop in U.

Rest te bewijzen dat elk van de knopen w_j een buur is van een knoop in U. Gezien e een waarheidstoekenning is die ϕ waar maakt, maakt deze in elke clausule C_j een van de drie literals waar. Als deze literal een zuivere variabele is, zeg x_i, dan is w_j in $T(\phi)$ verbonden met v_i^t, die per constructie bevat is in U omdat $e(x_i) = \text{true}$. Is deze literal een negatieve variabele, zeg $\neg x_i$, dan is w_j verbonden met v_i^f, die per constructie bevat is in U, omdat $e(x_i) = \text{false}$.

Iedere w_j is dus ook verbonden met U, dus U is inderdaad een dominating set in $T(\phi)$, dus $T(\phi)$ is een ja-instantie van DS.

c. Merk op dat iedere knoop v_i^0 alleen verbonden is met v_i^t en v_i^f. Om te zorgen dat iedere v_i^0 bevat is in U of verbonden is aan een knoop in U, is het dus noodzakelijk om uit ieder drietal $\{v_i^0, v_i^t, v_i^f\}$ minstens één knoop in U op te nemen.

In totaal mogen we niet meer dan n knopen kiezen voor U. Gezien er ook n van deze drietallen $\{v_i^0, v_i^t, v_i^f\}$ zijn, en we uit elk drietal een knoop moeten kiezen, mogen we ook niet méér dan één knoop uit een drietal kiezen, noch knopen w_j. De dominating set U moet dus precies bestaan uit één knoop uit elk drietal.

d. Stel dat $T(\phi)$ een ja-instantie is voor DS, en laat U een dominating set zijn voor $T(\phi)$. Uit opgave c volgt dat U precies bestaat uit één knoop uit elk drietal $\{v_i^0, v_i^t, v_i^f\}$. Neem als waarheidstoekenning voor ϕ nu $e(x_i) = \text{true}$ als $v_i^0 \in U$ of $v_i^t \in U$, en $e(x_i) = \text{false}$ als $v_i^f \in U$. We tonen aan dat e een waarheidstoekenning is die ϕ waar maakt.

Per constructie is iedere knoop w_j verbonden met een knoop in U. Gezien geen van de w_j verbonden is met een v_i^0, geldt dus dat iedere w_j verbonden is met een $v_i^t \in U$ of met een $v_i^f \in U$. In het eerste geval komt de variabele x_i in zuivere vorm voor in C_j, en maakt de toekenning $e(x_i) = \text{true}$ deze clausule dus waar. In het tweede geval komt x_i in negatieve vorm voor in C_j, en maakt de toekenning $e(x_i) = \text{false}$ deze clausule dus waar.

De waarheidstoekenning e maakt ϕ dus inderdaad waar, dus ϕ is een ja-instantie voor 3-SAT.

e.

1. Waar. Uit (b), (d) en $(*)$ volgt dat ϕ een polynomiale reductie is van 3-SAT naar DS, wat precies de definitie is van 3-SAT $\leqslant_{\text{P}}$ DS.

2. Waar. 3-SAT is NP-hard, wat betekent dat alle problemen in $\mathcal{NP}$ gereduceerd kunnen worden naar 3-SAT. In (a), hebben we bewezen dat DS $\in \mathcal{NP}$, dus in het bijzonder geldt DS $\leqslant_{\text{P}}$ 3-SAT.

3. Waar. 3-SAT is NP-hard, dus uit de transitiviteit van $\leqslant_{\text{P}}$ en uitspraak 1 volgt dat ook DS een NP-hard probleem is. Uit (a) weten we dat DS $\in \mathcal{NP}$, dus per definitie geldt dan dat DS $\in \mathcal{NPC}$

4. Waar. We weten dat $\mathcal{P} \subseteq \mathcal{NP}$, dus ook $P \in \mathcal{NP}$. 3-SAT is NP-hard, wat betekent dat alle problemen in NP gereduceerd kunnen worden naar 3-SAT. In het bijzonder dus ook $P \leqslant_{\text{P}}$ 3-SAT.

5. Zeer waarschijnlijk onwaar. Als dit het geval was, zou uit de transitiviteit van $\leqslant_{\text{P}}$ volgen dat ook P een NP-hard probleem is. Uit $P \in \mathcal{P}$ zou dan geconcludeerd kunnen worden dat $\mathcal{P} = \mathcal{NP}$.