Tentamen Complexiteit
Donderdag 20 juni 2024, 9.00 – 12.00 uur

Geef een *duidelijke* toelichting bij al je antwoorden! Veel succes!

Opgave 1. (25 punten)
Gegeven een array $A = A[1], A[2], \ldots, A[n]$ dat n verschillende getallen bevat ($n = 2^k$ voor zekere $k \geq 3$). We willen de grootste en de op-een-na-grootste waarde uit A vinden. We zoeken dus het paar (p, q) waarvoor geldt dat $A[p]$ de grootste waarde uit A is en $A[q]$ de op-een-na-grootste.

a. (5 punten)
Leg uit hoe de toernooimethode werkt voor algemene $n = 2^k$ en leg uit hoeveel arrayvergelijkingen deze doet. Illustreer je uitleg ten slotte met een voorbeeld waarvoor $n = 16$.

We nemen vanaf hier aan dat het array A $\frac{n}{8}$-gesorteerd[1] is. Een voorbeeld met $n = 16$: $5, 4, 9, 11, 14, 17, 15, 20, 25, 22, 27, 30, 31, 35, 32, 38$. Hiervoor is $(p, q) = (16, 14)$.

b. (4 punten)
Gegeven een beslissingsboom corresponderend met een algoritme dat gebaseerd is op arrayvergelijkingen. In termen van het algoritme, wat stelt een pad van de wortel naar een blad voor, wat is de betekenis van de lengte van zo'n pad, wat kun je zeggen over de bladeren en wat stelt de hoogte van zo'n boom voor?
c. (8 punten)
Bewijs met behulp van een *beslissingsboomargument* dat elk algoritme voor bovenstaand probleem dat is gebaseerd op arrayvergelijkingen, voor de bekeken soort invoerrijtjes ten minste $2 \lg n - 6$ arrayvergelijkingen moet doen in de worst case. Formuleer je bewijs zorgvuldig, maak gebruik van **b** en geef aan welke stellingen/eigenschappen je gebruikt.
d. (3 punten)
Op college is verteld dat de toernooimethode (zie **a**) optimaal is. Betekent dit dat de ondergrens uit **c** niet scherp is? Motiveer je antwoord.
e. (5 punten)
Door de vorm van het invoerarray te gebruiken kun je de grootste en de op-een-na-grootste waarde uit het array vinden met echt minder vergelijkingen dan de "gewone" toernooimethode uit **a**. Geef zo'n algoritme dat (i) handig gebruik maakt van de toernooimethode en (ii) als hoogste orde term voor het aantal arrayvergelijkingen maximaal $\frac{n}{4}$ heeft.

Opgave 2. (20 punten)
Gegeven is een array $A = A[1], A[2], \ldots, A[n]$, dat verschillende getallen bevat en waarbij $n = 4^k$ voor een geheel getal $k \geq 0$ (*). We gaan het array oplopend sorteren met behulp van een aangepaste versie van Mergesort.
We verdelen het array eerst in 4 stukken van elk $\frac{n}{4}$ elementen:

[1] Een array heet ℓ-gesorteerd als geldt: $A[i] \leq A[i + \ell]$ voor elke $i = 1, 2, \ldots, n - \ell$.

I	II	III	IV

Vervolgens sorteren we recursief elk deelarray van lengte $\frac{n}{4}$ en ten slotte voegen we de vier gesorteerde delen als volgt samen met behulp van het bekende Merge-algoritme. Eerst mergen we I en II; vervolgens mergen we het resultaat met III. Nu zijn de linker $\frac{3n}{4}$ elementen gesorteerd en mergen we deze ten slotte met IV.

a. (5 punten) Neem aan dat de vier deelarrays I, II, III en IV reeds oplopend gesorteerd zijn. Hoeveel arrayvergelijkijkingen kosten de drie merge-acties samen in de best case en hoe moet het array er uitzien om dit aantal te bereiken? Dezelfde vraag voor de worst case. Leg je antwoord uit.

We bekijken nu weer een willekeurig array A dat voldoet aan (*). We sorteren dit recursief met behulp van bovenbeschreven versie van Mergesort. Het aantal arrayvergelijkingen dat dit algoritme doet in de worst case noemen we $M(n)$. Dan voldoet $M(n)$ aan:

$$M(n) = \begin{cases} 0 & n = 1 \\ 4M(\frac{n}{4}) + \frac{9}{4}n - 3 & n \geq 4, n = 4^k \quad (\text{dus } k = \log_4 n) \end{cases}$$

b. (10 punten)
Los de recurrente betrekking exact op door deze herhaald in zichzelf in te vullen (substitutiemethode). Schrijf $M(n)$ als functie van n. Bewijs vervolgens met behulp van volledige inductie dat de aldus gevonden oplossing inderdaad voldoet.
Je mag gebruiken dat $\sum_{i=0}^{\ell-1} 4^i = \frac{1}{3}(4^\ell - 1)$.

Onderdeel **c** staat los van **a** en **b**
c. (5 punten)
Veronderstel nu dat het array A bestaat uit 4 *aflopend* gesorteerde deelarrays van elk $\frac{n}{4}$ elementen, dus als in het plaatje, maar met I t/m IV aflopend gesorteerd.
Bereken een ondergrens op het aantal arayvergelijkingen dat Insertion sort ten minste moet doen, door het minimale aantal inversies van A te bepalen. Geef aan welke stelling(en)/eigenschappen je gebruikt.

Opgave 3. (25 punten)
Gegeven is een array $A = A[1], \ldots, A[n]$ dat $n \geq 2$ gehele positieve (> 0) getallen bevat. We bekijken een algoritme dat voor elke positie i telt hoeveel latere (aaneengesloten) deelrijtjes sommeren tot $A[i]$. Formeel: het algoritme berekent voor elke positie i het aantal posities j zodat $A[i] = A[j] + \ldots + A[j + k]$ voor zekere $j > i$ en $k \geq 0$.

Voorbeeld: in de rij $7, 8, 4, 3, 1, 4, 2, 5$ kan dit voor de elementen op respectievelijk $3, 2, 2, 0, 0, 0, 0$ manieren. Zo is 7 te schrijven als $4 + 3$ (positie 3), $1 + 4 + 2$ (positie 5) en $2 + 5$ (positie 7).

Het algoritme berekent dit door voor elke i een zogeheten *sliding window* toe te passen: het houdt de som bij van het deelrijtje tussen posities *links* en *rechts*, en voegt rechts nieuwe elementen toe of verwijdert ze links naargelang de huidige som groter of kleiner moet worden. Zo 'schuift' het deel van het array dat door het algoritme wordt bekeken telkens op. Wanneer het huidige deelrijtje sommeert tot $A[i]$, wordt dit genoteerd in het hulparray B. Het

algoritme stopt met opschuiven wanneer het rechts een nieuw element zou moeten toevoegen en dit niet langer kan.
De arrayvergelijking $som \geq A[i]$ op regel 4 is maatgevend voor de complexiteit.

```
1  i := 1; B[n] = 0;
2  while i < n do
3  |  B[i] = 0; links := i + 1; rechts := i + 1; som := A[i + 1];
4  |  while som ≥ A[i] or rechts < n do          // let op: hier staat or!
5  |  |  if som = A[i] then
6  |  |  |  B[i] := B[i] + 1;
7  |  |  fi
8  |  |  if som ≥ A[i] then
9  |  |  |  som := som − A[links]; links := links + 1;
10 |  |  else
11 |  |  |  rechts := rechts + 1; som := som + A[rechts];
12 |  |  fi
13 |  od
14 |  i := i + 1;
15 od
16 return B;
```

a. (5 punten)
Voer het algoritme uit voor $i = 1$ en $i = 2$ op het rijtje $7, 1, 3, 3, 10, 1, 2, 2$. Hoe vaak wordt de arrayvergelijking op regel 4 gedaan? Geef alle tussenwaardes van $B[i]$, *links*, *rechts* en *som*.
b. (10 punten)
Hoe vaak doet dit algoritme de arrayvergelijking op regel 4 in de *best case*? Geeft een exact antwoord en beschrijf *alle* bijbehorende invoerrijtjes. Leg op basis van het algoritme uit hoe je aan je antwoord komt.
c. (10 punten)
Hoe vaak doet dit algoritme de arrayvergelijking op regel 4 in de *worst case*? Geeft een exact antwoord en beschrijf *alle* bijbehorende invoerrijtjes. Leg op basis van het algoritme uit hoe je aan je antwoord komt.

Opgave 4. (30 punten)
We bekijken de volgende twee beslissingsproblemen:

3SAT: Gegeven een logische formule ϕ in 3-CNF. **Vraag:** bestaat er een waardering van de in ϕ voorkomende logische variabelen x_i die ϕ True maakt?
Definitie. Een logische formule ϕ staat in 3-CNF als ϕ een conjunctie ($\wedge$) van clausules ($\vee$) is, waarbij *elke clausule* bestaat uit *drie verschillende literals*[2].

1-in-3-SAT: Gegeven een logische formule ϕ in 3-CNF. **Vraag:** bestaat er een waardering van de in ϕ voorkomende logische variabelen x_i die precies één literal per clausule waar maakt (en dus heel ϕ True maakt).

Voorbeeld: De formule $(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z)$ is een ja-instantie voor 1-in-3-SAT; $(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$ is een nee-instantie.

[2]Een literal is een variabele x_i of zijn negatie $\neg x_i$

a. (8 punten)
Toon aan dat 1-in-3-SAT $\in \mathcal{NP}$ door een *niet-deterministisch polynomiaal* algoritme voor 1-in-3-SAT te geven. Het algoritme heeft dus als invoer een logische formule ϕ in 3-CNF en moet "ja" opleveren dan en slechts dan als $x = \phi$ een ja-instantie is voor 1-in-3-SAT (&). Je mag aannemen dat het aantal verschillende logische variabelen in ϕ bekend is en gelijk is aan n.
Leg uit wat in elke fase van het algoritme gebeurt en hoeveel stappen dat kost. Leg ook uit waarom jouw algoritme aan (&) voldoet en waarom het polynomiaal is in $|x|$.

We transformeren instanties van 3SAT naar instanties van 1-in-3-SAT.
Gegeven een logische formule ϕ in 3-CNF. We construeren uit ϕ een logische formule $T(\phi)$. Voor elke clausule $(\ell_1 \vee \ell_2 \vee \ell_3)$ van ϕ (met de ℓ_i verschillende literals) introduceren we 4 nieuwe logische variabelen a, b, c en d. We beelden nu de clausule $C = (\ell_1 \vee \ell_2 \vee \ell_3)$ af op $T(C) = (\neg\ell_1 \vee a \vee b) \wedge (b \vee \ell_2 \vee c) \wedge (c \vee d \vee \neg\ell_3)$. Voor de formule $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ wordt de transformatie T dan gedefinieerd door $T(\phi) = T(C_1) \wedge T(C_2) \wedge \cdots \wedge T(C_m)$ (voor elke C_i nieuwe, frisse variabelen a_i, b_i, c_i en d_i).
Het is duidelijk dat de constructie van $T(\phi)$ uit ϕ polynomiaal is. Samen met **b** en **c** volgt dan dat 3SAT $\leq_{\mathrm{P}}$ 1-in-3-SAT.

Voorbeeld: $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_1 \vee x_4)$ wordt afgebeeld op: $(\neg x_1 \vee a_1 \vee b_1) \wedge$
$(b_1 \vee x_2 \vee c_1) \wedge (c_1 \vee d_1 \vee x_3) \wedge (x_2 \vee a_2 \vee b_2) \wedge (b_2 \vee \neg x_1 \vee c_2) \wedge (c_2 \vee d_2 \vee \neg x_4)$

b. (5 punten)
Toon aan: ϕ is een ja-instantie van 3SAT $\Longrightarrow$ $T(\phi)$ is een ja-instantie van 1-in-3-SAT. Formuleer daartoe eerst wat het betekent als ϕ resp. $T(\phi)$ een ja-instantie is.
Hint: kijk naar één clausule en onderscheid gevallen.
c. (5 punten)
Toon aan: $T(\phi)$ is een ja-instantie van 1-in-3-SAT $\Longrightarrow$ ϕ is een ja-instantie van 3SAT.
d. (2 punten)
Wanneer is een beslissingsprobleem Q NP-hard? En wanneer NP-volledig? (De definitie van $\mathcal{NP}$ hoeft niet te worden gegeven.)

In vraag **e** mag je gebruiken dat (1) 3SAT $\in \mathcal{NPC}$, (2) 1-in-3-SAT $\in \mathcal{NP}$ en
(3) 3SAT $\leq_{\mathrm{P}}$ 1-in-3-SAT.

e. (10 punten)
Beantwoord de volgende vragen en geef bij elk antwoord een *duidelijke uitleg.*
(i) Waarom geldt ook dat 1-in-3-SAT $\leq_{\mathrm{P}}$ 3SAT?
(ii) Volgt uit (1), (2) en (3) dat 1-in-3-SAT $\in \mathcal{NPC}$ of is daar de omgekeerde reductie uit (i) voor nodig?

Definitie: een logische formule ϕ staat in disjunctieve normaalvorm (DNF) als ϕ een disjunctie ($\vee$) van conjuncties ($\wedge$) is.

(iii) Iemand beweert een polynomiaal algoritme te hebben bedacht dat een formule in 3-CNF omzet in een equivalente formule in DNF. Waarom is dat algoritme hoogstwaarschijnlijk fout?

EINDE