Examen Datastructuren en Algoritmen II

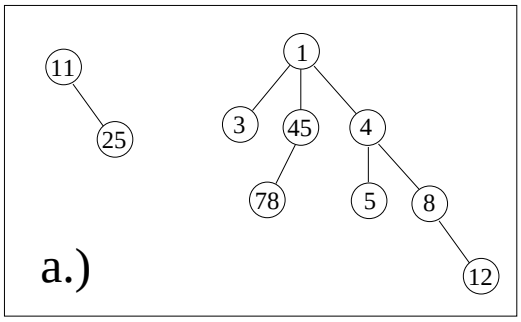
Naam : .....

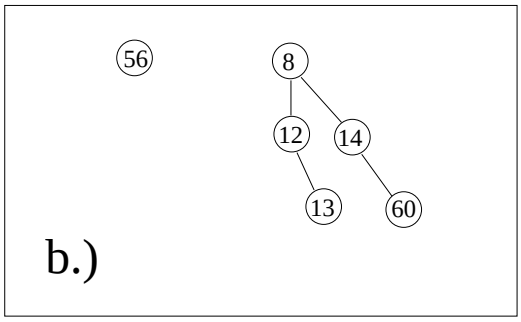
1. **Binomiale wachtlijnen:** (3 pt)

   a.) Mag een binomiale wachtlijn dezelfde sleutel meerdere keren bevatten ? Wat is het effect op de efficiëntie of juistheid van de in de les geziene bewerkingen als het anders was?

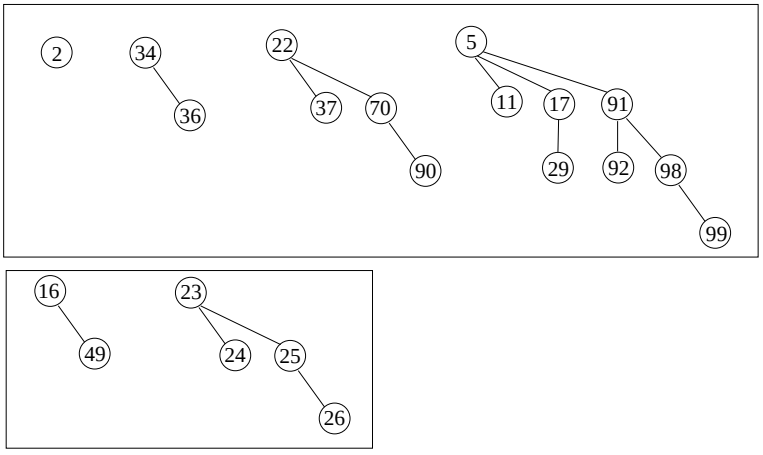
   b.) Vergelijk de efficiëntie van binomiale wachtlijnen met die van een *gewone* binaire hoop zoals gezien in AD I. Zijn er bewerkingen waar de complexiteit verschillend is als je naar de O() notatie kijkt?

   c.) Schrijf voor elk van de twee datastructuren of zij wel of niet een binomiale wachtlijn is. Indien niet geef aan waarom niet.





d.) Merge de twee volgende binomiale wachtlijnen. Toon voldoende tussenstappen die laten zien wat er gebeurt.



2. **Miller-Rabin** (3 pt)

Gegeven een getal n waarvoor wij willen testen of het priem is en een basis b.

a.) Beschrijf de twee gevallen wanneer de Miller-Rabin test zegt dat een getal geen priemgetal is. Waarop zijn deze beslissingen gebaseerd?

b.) Stel dat het resultaat van de Miller-Rabin test is "**geen** priemgetal". Hoe groot is de kans dat dit antwoord juist is?

c.) Pas de Miller-Rabin test toe op $n = 25$ en $b = 18$.

d.) Pas de Miller-Rabin test toe op $n = 21$ en $b = 8$.

3. **Gebalanceerde zoekbomen:** (2 pt)

In delen a.), b.) en c.) is de O() notatie voldoende.
Lees de oefening zorgvuldig – er is een verschil tussen *de kost van n bewerkingen* en *de kost per bewerking als er n bewerkingen gedaan worden*.

a.) Wat zijn in het slechtste geval de kosten van *n* toevoegingen op een initieel lege boom die nooit geherbalanceerd wordt? (Geen bewijs vereist.)

b.) Wat zijn in het slechtste geval de kosten van *n* toevoegingen op een initieel lege rood-zwart-boom? (Geen bewijs vereist.)

c.) 2-3-bomen hebben in het **slechtste** geval de diepte van een perfect gebalanceerde binaire boom – dus is de diepte in bijna alle gevallen veel beter dan die van AVL-bomen en rood-zwart-bomen. Waarom zijn AVL-bomen en rood-zwart-bomen toch interessant om te gebruiken?

d.) Geef de definitie van een 2-3-boom.

4. **Een probleem vertalen:** (2 pt)

   Een bedrijf heeft van een klant n werkstukken gekregen die met een frees bewerkt moeten worden. Voor elk werkstuk w_i is het aantal uren u_i dat nodig is al op voorhand gekend (voor verschillende i kunnen de u_i ook verschillen). Maar deze tijden zijn alleen maar geldig als het werkstuk het eerste is dat op een machine wordt bewerkt. Als voordat werkstuk w_i wordt bewerkt al een ander werkstuk w_j op dezelfde machine werd bewerkt is het niet nodig de machine opnieuw met werktuigen uit te rusten, dus is dan de tijd die nodig is maar $u_i - c$ voor een gegeven constante c.

   Het is belangrijk dat alle werkstukken na ten hoogste twee dagen klaar zijn, maar natuurlijk wil het bedrijf zo weinig mogelijk frezen gebruiken.

   Eén manier om dit probleem op te lossen, is het te vertalen naar een gekend probleem waarvoor al algoritmen bestaan. Geef zo'n vertaling naar een gekend probleem. Je moet geen algoritmen geven die het gekende probleem oplossen.

5. **Het inpakprobleem:** (2 pt)

   Gegeven een arbitraire reeks van gewichten $g_1, \ldots, g_n$.

   Toon aan of geef een tegenvoorbeeld:

   First fit: Gegeven een verdeling V van de gewichten op vrachtwagens met het kleinst mogelijke aantal vrachtwagens. Er is altijd een herordening $g_{i_1}, \ldots, g_{i_n}$ van deze reeks zodat als first-fit op de herordende reeks wordt toegepast het resultaat is dat de gewichten op dezelfde vrachtwagens geplaats worden dan door V.

   Best fit: Er is altijd een herordening $g_{i_1}, \ldots, g_{i_n}$ van deze reeks zodat als best-fit op de herordende reeks wordt toegepast het resultaat een verdeling met het kleinst mogelijke aantal vrachtwagens is.

Grenzen: Stel dat een optimale oplossing m vrachtwagens gebruikt voor de reeks $g_1, \ldots, g_n$ en dat $g_{i_1}, \ldots, g_{i_n}$ een arbitraire herordening van deze gewichten is. Geef een bovengrens voor het aantal vrachtwagens dat first-fit (**niet** first-fit-dalend) gebruikt als het op $g_{i_1}, \ldots, g_{i_n}$ wordt toegepast. Geef uitleg. Stellingen uit de les mogen hier natuurlijk gebruikt worden zonder ze te bewijzen.

6. **Dynamisch programmeren en gretige algoritmen:** (4 pt)

Stel dat wij voor een reeks van sleutels $s_1 < s_2 < \ldots < s_n$ al op voorhand weten hoe vaak de sleutels opgezocht gaan worden. Sleutel i wordt g_i keer opgezocht. Het doel is een binaire zoekboom te vinden zodat de opzoekingen zo efficiënt als mogelijk gebeuren – dus: waar zo weinig mogelijk toppen bezocht moeten worden. Als op de weg van de wortel naar sleutel i precies $d(i)$ sleutels zitten, is de kost van alle opzoekingen

$$\sum_{i=1}^{n} (d(i) * g(i))$$

a.) Geef een gretig algoritme voor dit probleem en toon aan dat jouw gretig algoritme niet altijd de beste oplossing vindt. Is de kost van de gevonden oplossing altijd maar een constante factor slechter dan de beste oplossing?

b.) Geef een algoritme met dynamisch programmeren dat de kost van een optimale oplossing bepaalt.

Een tip die gebruikt mag worden (maar niet moet): Stel dat voor $a \leq b$ de boom $B(a,b)$ de boom met de minimale kost voor sleutels $s_a, s_{a+1}, \ldots, s_b$ is en $B(a,b) = \emptyset$ als $b < a$. Dan is de boom met de kleinste kost **en** sleutel c als wortel de boom die $B(1,c-1)$ als linkerdeelboom en $B(c+1,n)$ als rechterdeelboom heeft. De kost van deze boom is dan $g_c + kost(B(1,c-1)) + kost(B(c+1,n)) + a(1,c-1) + a(c+1,n)$ waarbij

$$a(i,j) = \sum_{k=i}^{j} g_k$$

dus gewoon het aantal keren dat sleutels in deze deelboom worden opgezocht.

NOG NIET OMDRAAIEN !