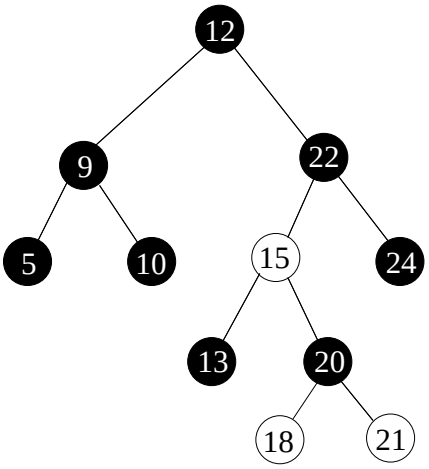
Examen Datastructuren en Algoritmen II

Naam : .....

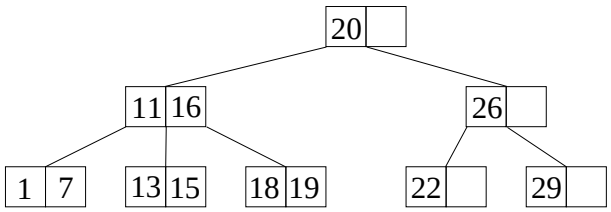
- Lees de hele oefening zorgvuldig voordat je begint ze op te lossen!
  Als je niet goed verstaat wat de vraag of taak is, vraag het aan de lesgever! Voor oefeningen die fout verstaan zijn, kunnen geen punten gegeven worden!
- Schrijf leesbaar. Oplossingen die niet leesbaar zijn, kunnen ook niet beoordeeld worden.
- Als technieken toegepast moeten worden, toon altijd voldoende tussenstappen om te kunnen zien wat er gebeurt en dat de technieken goed verstaan zijn.
- Stellingen uit de les mogen natuurlijk altijd gebruikt worden zonder dat het bewijs opnieuw gegeven moet worden (behalve in gevallen waar het expliciet anders staat)!
- Geef alleen dan een antwoord als je denkt dat je de oplossing kent. Verspil geen tijd met de poging gewoon lange teksten te schrijven waarin zekere sleutelwoorden opduiken – zoals dat vaak geprobeerd wordt. Dergelijke oplossingen halen nooit punten en voor bijzonder slechte oplossingen worden punten afgetrokken!

1. **Zoekbomen 2.5 pt**

   - Voeg sleutel 19 toe aan de volgende rood-zwart boom. Voor het herbalanceren gebruik – waar mogelijk – de manier die het toelaat, soms vroegtijdig met het herbalanceren te stoppen omdat de wortel van de vervangende boom zwart gekleurd kan worden.



- Verwijder sleutel 22 uit de volgende 2-3 boom



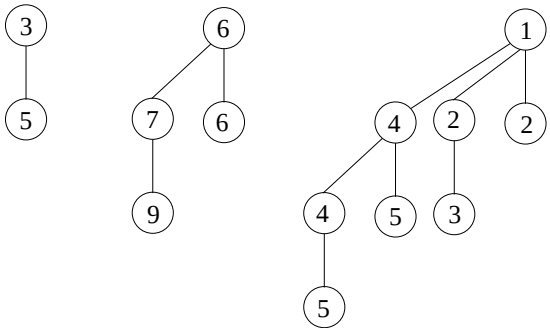
- Wij weten dat in een reeks van n bewerkingen op een initieel lege semi-splay boom een bewerking kost $n - 1$ kan hebben (als kost beschouwen wij het aantal bezochte toppen). In het volgende veronderstellen wij dat het aantal bewerkingen een veelvoud van 5 is om niet te moeten afronden:

  Bewijs: In een reeks van $5 * n$ bewerkingen op een initieel lege semi-splay boom zijn er geen twee bewerkingen die beide een kost van ten minste $4 * n$ hebben.

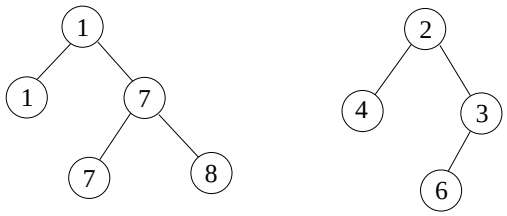
  Resultaten uit de les en de oefeningen mogen natuurlijk gebruikt worden.

2. **Heaps 2.5 pt**

- Verwijder de kleinste sleutel uit de volgende binomiale prioriteitswachtlijn.



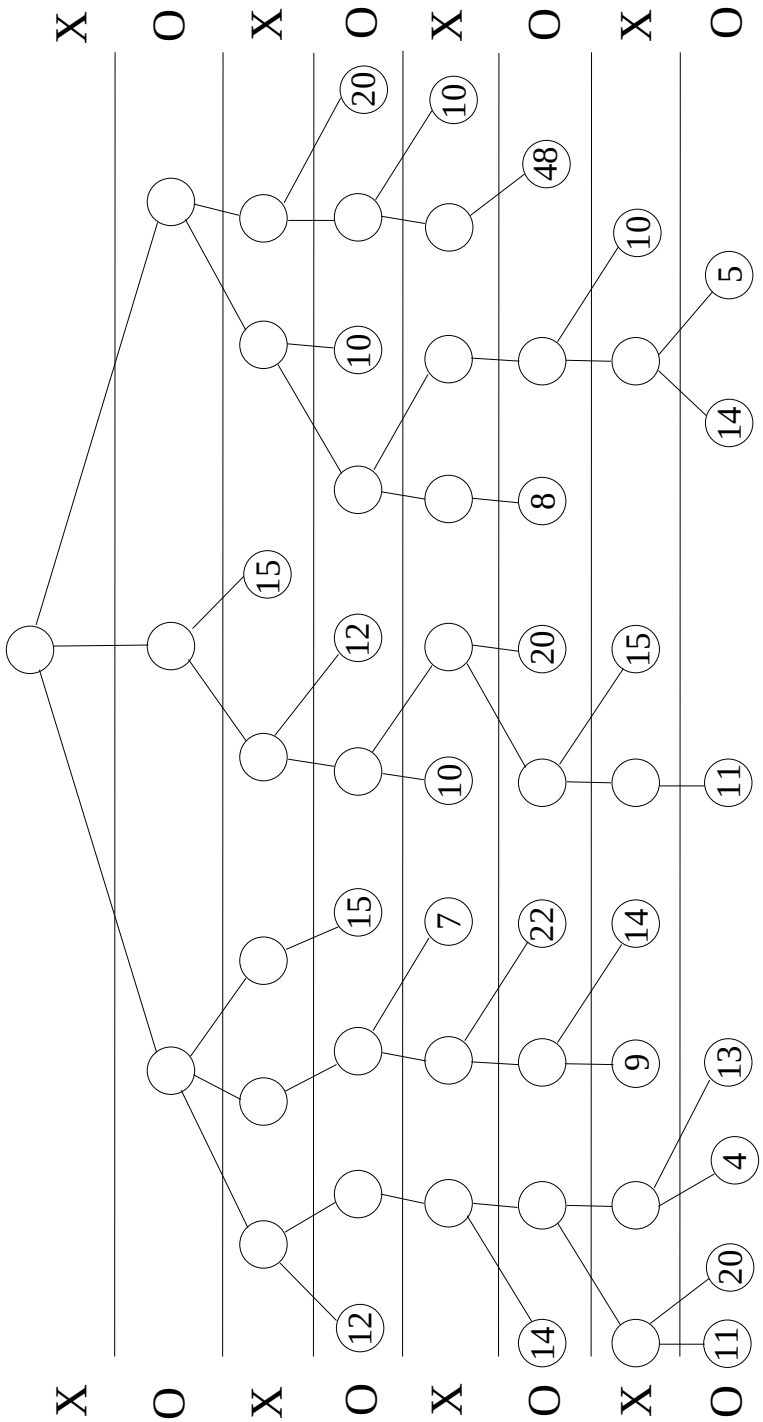
- Merge de volgende twee skew heaps met de recursieve skew-merge-bewerking.



- Je krijgt drie rijen van getallen die je aan een oorspronkelijk lege heap moet toevoegen. Elke van deze rijen bevat zeer veel getallen. Schrijf voor elke rij van welke heap die je gezien hebt je zou verwachten dat die het snelst opgebouwd kan worden. Geef ook jouw redenen voor deze keuze, maar echte bewijzen zijn hier niet gevraagd. Je moet elk getal toevoegen zodra je het hebt, je mag dus niet eerst wachten totdat je alle getallen kent.
  - Een dalende rij van getallen.
  - Een toevallig verdeelde rij van getallen.
  - Een stijgende rij van getallen.

3. **De waarde van een spel 1 pt**

- Pas α–β–snoeien op de volgende spelboom toe om de waarde van het spel te berekenen. Evalueer altijd **eerst de rechtertakken**. Schrijf gebruikte grenzen **aan** de toppen en de teruggegeven waarden **in** de toppen. Plaats streepjes door de bogen die naar deelgrafen leiden die je niet evalueert omdat je snoeit – en alleen door deze bogen.



4. **Geamortiseerde complexiteit 2 pt**

   Je wilt getallen aan een lijst toevoegen, zodat ze ten minste *een beetje* gesorteerd zijn – dat betekent: als er k getallen $g_1, \ldots, g_k$ in deze volgorde in de array zitten voeg je het volgende getal g in het eerste vakje links van de al geplaatste getallen toe als $g \leq g_{\lceil k/2 \rceil}$ en anders in het eerste vakje rechts ervan. Je begint met een array met 5 plaatsen en plaatst het eerste element in het midden. Deze eerste array noem je *nieuw*.

   Natuurlijk kan het als je links of rechts een element wilt toevoegen gebeuren dat er geen plaats meer is. Wat je dan doet wordt alleen beschreven als je iets links wilt toevoegen – voor rechts is de werking analoog. Als de array nieuw is en je wilt iets links toevoegen, maar er is geen ruimte (maar rechts wel), dan verschuif je alle elementen zodat ze opnieuw gecentreerd staan (dat is: rechts en links zijn op ten hoogste 1 na even veel vrije plaatsen en als het aantal vrije plaatsen oneven is dan is er links 1 meer) en voeg je het element toe. Dan noem je de array *oud*.
   Als de array van lengte l oud is of helemaal volzit, dan kies je een nieuwe array (je noemt het ook opnieuw *nieuw*) van lengte $2 * l + 1$ en je plaatst de elementen uit de vorige array gecentreerd.

   De kost van een toevoegbewerking als de elementen noch verschoven noch naar een nieuwe array verplaatst moeten worden is 1. Als de elementen eerst verschoven moeten worden is de kost van de toevoegbewerking $m + 1$ met m het aantal elementen dat verschoven wordt. Als de array gedubbeld wordt is de kost $2 * l + 1$ met l het aantal plaatsen in de oude array.

   - Wat is de maximale kost van een bewerking in een reeks van n toevoegbewerkingen op een initieel lege array. De $O()$ notatie is voldoende. Geef uitleg en ook een reeks en een bewerking zodat duidelijk is dat jouw grens een goede grens is.

- Wat is de geamortiseerde kost van één bewerking in een reeks van n toevoegbewerkingen op een initieel lege array. De $O()$ notatie is voldoende. Je mag vrij kiezen op welke manier je de grens bewijst, maar het moet wel een goede grens zijn.

5. **Dynamisch programmeren 2 pt**

   Je hebt n dozen $d_1, \ldots, d_n$. Doos i heeft (in cm) hoogte h_i en grondvlak $l_i \times b_i$. Je kan doos d_i op doos d_j plaatsen als de dimensies van het grondvlak allebei kleiner zijn, waarbij je l en b natuurlijk kan verwisselen door de doos te draaien. Je kan dus een doos met $l = 4, b = 7$ op een doos met $l = 8, b = 5$ plaatsen, maar niet op een doos met $l = 7, b = 7$. De bedoeling is nu te berekenen hoe hoog de hoogste toren is die je met deze dozen kan bouwen. Geef een algoritme met dynamisch programmeren, maar dat betekent niet dat je niet misschien eerst de volgorde van de dozen mag wijzigen...

   Geef de pseudocode **en** voldoende uitleg om de pseudocode te verstaan en om aan te tonen dat het algoritme juist is.

6. **Online algoritmen 2 pt**

- In de les hebben wij bewezen dat voor elk online inpakalgoritme en elk getal k een rij van gewichten bestaat, zodat het optimum $m \geq k$ vrachtwagens is en het algoritme ten minste $\frac{4}{3}m$ vrachtwagens gebruikt. Dat is voor het geval dat je helemaal niets weet over het algoritme – behalve dat het online is.

  Stel nu dat je alleen uitspraak wilt doen over online inpakalgoritmen die nooit een nieuwe vrachtwagen gebruiken als ze een gewicht ook op een al gebruikte vrachtwagen kunnen plaatsen. Laat ons deze online algoritmen *conservatief* noemen. Wij zullen de stelling ook iets sterker formuleren. Wij willen een stelling

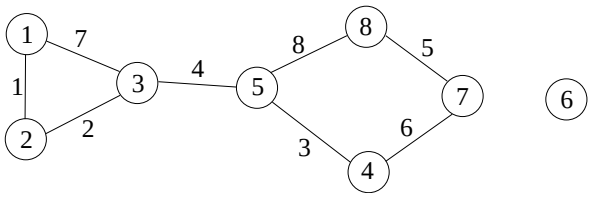
  Stelling: Voor elk conservatief online inpakalgoritme en elk getal k bestaat een rij van gewichten, zodat het optimum $m \geq k$ vrachtwagens is en het algoritme ten minste $c \cdot m$ vrachtwagens gebruikt.

  – Bewijs deze stelling voor een constante $c > 1.6$

- Toon aan dat deze stelling niet geldt voor $c > 1.8$

7. **Verzamelingen 2 pt**

- In de les werd gezegd dat samenhangscomponenten van een graaf de door de bogen geïnduceerde equivalentieklassen van toppen zijn. Bereken de samenhangscomponenten van de volgende graaf door union-find met *union by size* en *path compression* toe te passen. Pas de door de bogen gegeven relaties in de volgorde toe die door de getallen aan de bogen gegeven wordt (de eerste relatie is dus $1 \equiv 2$ en de laatste $5 \equiv 8$). Als door *union by size* niet vastgelegd wordt, welke wortel bij het verenigen van twee bomen de nieuwe wortel wordt, neem altijd de wortel die kleiner is.



- Een graaf met toppenverzameling $\{0,\ldots,n\}$ met $n < 64$ is gegeven door de integer array `g[]`, waarbij voor $0 \leq i \leq n$ de integer $g[i]$ de verzameling van buren van top i als bitvector bevat. Geef efficiënte uitdrukkingen die *true* opleveren als en slechts als
  - de toppen i, j, k een driehoek in de graaf vormen
  - de toppen i en j ten hoogste één gemeenschappelijke buur hebben
  - de top i een buur van alle andere toppen in de graaf is

8. **Gerandomiseerde algoritmen 2 pt**

- Pas de priemgetallentest van Miller-Rabin toe om te toetsen of 27 een priemgetal is. Kies 7 als *toevallig gekozen* waarde tussen 0 en 27.

- Gegeven een rij van $3 * n$ verschillende getallen $x_1, \ldots, x_{3*n}$. Deze rij is niet gesorteerd, maar je zoekt een getal x_i zodat er ten minste n kleinere en ten minste n grotere getallen zijn.

  Geef een Las Vegas algoritme dat in tijd $O(n)$ draait en met kans $c > \frac{1}{2}$ (waarbij c een van n onafhankelijke constante is) ten laatste na twee toepassingen een dergelijk getal gevonden heeft. Bewijs dat jouw algoritme deze eigenschap heeft.

NOG NIET OMDRAAIEN !