Opgave 1 (15%)

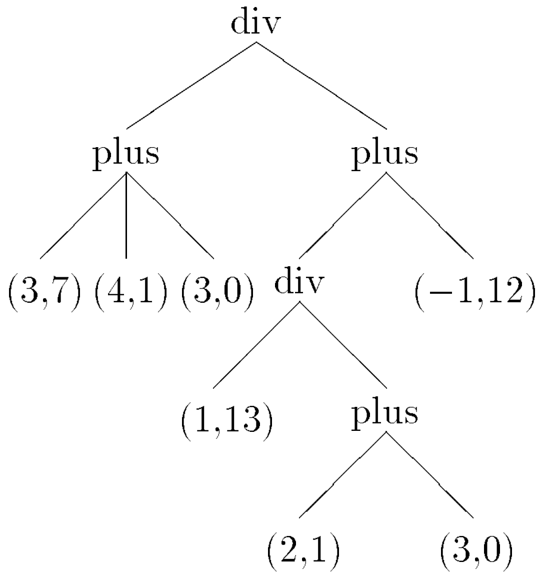
En *rational* funktion er enten et polynomium, en sum af eller en kvotient mellem rationale funktioner. De kan repræsenteres som værdier af følgende rekursive TRINE type

> **Type** Rat = **Sum**(sim: Simpel, plus: Liste, div: Par)
> **Type** Simpel = **Prod**(k, n: Int)
> **Type** Liste = **List**(Rat)
> **Type** Par = **Prod**(v, h: Rat)

Fx vil den rationale funktion

$$\frac{3x^7 + 4x + 3}{\dfrac{x^{13}}{2x+3} - x^{12}}$$

blive repræsenteret af Rat-værdien, der kan skitseres som følger



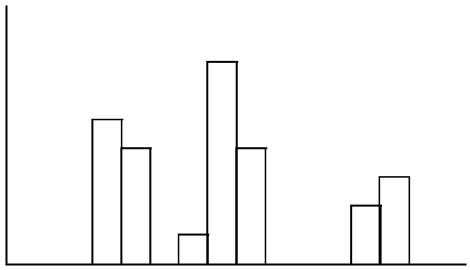
> Skriv en TRINE værdiprocedure
>
> **Proc** Eval[R: Rat](x: Int) → (Int)
>
> der beregner værdien af den rationale funktion R i punktet x. Der lægges vægt på, at besvarelsen er læselig, detaljeret og korrekt.

Opgave 2 (15%)

Der skal konstrueres en box Histo med følgende udseende

> **Box** Histo
>   **Type** H = ≪et histogram≫
>   **Proc** Init[h: H]
>   **Proc** Up[h: H](k, n: Int)
>   **Proc** Down[h: H](k, n: Int)
>   **Proc** Ask[h: H](k, n: Int) → (Bool)
>   **Proc** Area[h: H] → (Int)
> **end** Histo

som realiserer en datastruktur hvis værdier er ***histogrammer***, der ser ud som følger



Der er en ***søjle*** for hvert heltal $k \geq 0$, og hver søjle har en heltallig ***højde***. Proceduren Init giver et histogram hvor alle søjler har højde 0. Proceduren Up forøger højden af den k'te søjle med n. Proceduren Down mindsker højden af den k'te søjle med n; højden kan dog aldrig komme under 0. Proceduren Ask fortæller, om den k'te søjle har højde mindst n. Proceduren Area giver arealet af histogrammet.

a) Angiv en specifikation af proceduren Ask.

I det følgende lader vi |h| angive antallet af søjler med højde > 0 i histogrammet h.

b) Beskriv en realisation af typen H, så Init får tidskompleksitet $O(1)$, Up, Down og Ask får tidskompleksitet $O(\log \lvert h \rvert)$, og Area får tidskompleksitet $O(1)$.

Opgave 3 (15%)

Vi betragter en Real-vektor V, i hvilken alle elementer er ≥ 0. Det ***maksimale delprodukt*** af V er defineret ved

$$\text{maxprod}(V) = \max\{V.(j)V.(j{+}1)V.(j{+}2)\cdots V.(i{-}1) \mid 0 \leq j \leq i \leq |V|\}$$

Vi har fx, at for

$$\begin{aligned} V &= (5.7,\ 2.3,\ 0.4,\ 8.2,\ 2.7,\ 0.999) \\ W &= (0.5,\ 0.2,\ 0.87) \end{aligned}$$

så er $\text{maxprod}(V) = V.(3)V.(4) = 22.14$ og $\text{maxprod}(W) = 1$ (bemærk, at et "tomt" produkt har værdi 1). Det følgende er en skitse af en algoritme, der beregner det maksimale delprodukt.

Algoritme: Maksimalt Delprodukt
Stimulans: V: **List**(Real), $\forall i \in 0..|V| : V.(i) \geq 0$
Respons: $m = \text{maxprod}(V)$
Metode: $m, p, i := 1, 1, 0$
do $\{I\}$
  $i < |V| \rightarrow$
    $\ll$iterer$\gg$
    $i := i + 1$
od

Invarianten I siger, at

$$\begin{aligned} m &= \text{maxprod}(V(0..i)) \\ p &= \max\{V.(j)V.(j{+}1)V.(j{+}2)\cdots V.(i{-}1) \mid 0 \leq j \leq i\} \end{aligned}$$

Udfyld $\ll$iterer$\gg$, så algoritmen bliver gyldig og korrekt. Bevis, at algoritmen er korrekt.

Opgave 4 (15%)

I denne opgave behøver man ikke at skrive detaljerede programmer.

Betragt ***Knapsack*** problemet, beskrevet i ALGORITMISK PROBLEMLØSNINGSTEKNIK side 31.

Vi ser nu på en variation, hvor man har ***to*** rygsække, med kapaciteter henholdsvis C_1 og C_2. Givet et antal objekter $o_1, \ldots, o_n$ med størrelser $s_1, \ldots, s_n$ og værdier $v_1, \ldots, v_n$ skal man pakke de to rygsække på en sådan måde, at det samlede indhold har størst mulig værdi. Vi betegner med $MV(C_1, C_2)$ denne største værdi.

> **a)** Beskriv kort, hvordan den rekursive Knapsack algoritme på side 32 skal modificeres for at klare to rygsække.

Man kan opnå en stor effektivitetsforbedring ved at benytte dynamisk programmering.

> **b)** Beskriv kort, hvordan Tabel Knapsack algoritmen på side 33 skal modificeres for at klare to rygsække. Hvad bliver den forbedrede tidskompleksitet?

Opgave 5 (15%)

Betragt følgende TRINE program

```
Process P
    (+ Type A = List(List(B))
       Type B = Prod(x: C, y: Unit)
       Type C = Sum(z: A)
       Type D = List(B)

       Proc Q[a: List(D)]
           (+ Type E = List(List(C))
              Type C = Prod(x: Sum(z: E), y: Unit)

              Var e: E

              e := a
           +)
       end Q

       Var a: A

       a := List(List())
       Q[a]
    +)
end P
```

Vil det blive accepteret af oversætteren? Begrund dit svar.

Opgave 6 (15%)

Betragt følgende variation af del-og-kombiner algoritmen til flettesortering.

> **proc** Sorter[w: Vector]
>   **if** $|w| > 1 \ \rightarrow$
>   (+ **var** $w_1, w_2, \ldots, w_k$: Vector
> $w_1 := w(0..|w|/k)$
> $w_2 := w(|w|/k..2|w|/k)$
> $\vdots$
> $w_k := w((k-1)|w|/k..|w|)$
>   Sorter[w_1]
>   Sorter[w_2]
> $\vdots$
>   Sorter[w_k]
> $w := w_1$
> $w :=$ Flet[w, w_2]
> $\vdots$
> $w :=$ Flet[w, w_k]
>   +)
>   **fi**
> **end** Sorter

Et kald af Flet[v, w] returnerer fletningen af v og w i tid $O(|v| + |w|)$.

a) Angiv tidskompleksiteten for sorteringen af en vektor af længde n for et givet konstant k.

b) Hvad sker med tidskompleksiteten i det ekstreme tilfælde hvor $k = n$?

Opgave 7 (10%)

Betragt en liste af data med heltalsnøgler. Den kan fx beskrives som en værdi af følgende TRINE type.

> **Type** Element = **Prod**(nøgle: Int, data: Data)
> **Type** Liste = **List**(Element)

Sådanne lister kan sorteres på deres nøgle-værdier med de gængse sorteringsalgortimer. Bemærk, at disse algoritmer ikke kan referere til dataværdier.

Vi vil kalde en sådan algoritme for ***stabil***, hvis den lader data med samme nøgle-værdi beholde den samme indbyrdes rækkefølge i den sorterede liste som i den oprindelige liste.

Er følgende sorteringsalgoritmer stabile? Hvis ikke, kan man da let ændre dem, så de bliver det?

- Indsættelsessortering (PROGRAMMERINGSTEORI side 20)
- Udvalgssortering (PROGRAMMERINGSTEORI side 20)
- Quicksort (ALGORITMISK PROBLEMLØSNINGSTEKNIK side 8)
- Flettesortering (ALGORITMISK PROBLEMLØSNINGSTEKNIK side 12)