# DAT1 Sommereksamen 1991

Opgavesættet består af 8 uafhængige opgaver, der indgår med de angivne vægte i bedømmelsen.

## Opgave 1 (15%)

Betragt følgende rekursive TRINE type:

```
Type Logisk = Sum (konst: Bool,
                   negation: Logisk,
                   konjunktion, disjunktion: LogiskListe)
Type LogiskListe = List (Logisk)
```

hvis værdier kan opfattes som logiske udtryk. Negation tager et enkelt argument, hvorimod disjunktion og konjunktion tager et vilkårligt antal argumenter (inklusive nul). Værdien af en konstant er blot dens indhold. Værdien af en negation er true hvis værdien af dens argument er false, og false hvis værdien af dens argument er true. Værdien af en disjunktion er true hvis mindst et af dens argumenter har værdi true; ellers er værdien af disjunktionen false. Værdien af en konjunktion er false hvis mindst et af dens argumenter har værdi false; ellers er værdien af konjunktionen true. Vi har fx at værdien af

```
Logisk(konjunktion: LogiskListe(
                        Logisk(konst: true),
                        Logisk(disjunktion: LogiskListe())
                       )
      )
```

er false.

> Skriv en TRINE værdiprocedure
>
> **Proc** Afgør[L: Logisk] → (Bool)
>
> der returnerer værdien af det logiske udtryk L. Der lægges vægt på, at besvarelsen er detaljeret og korrekt.

## Opgave 2 (15%)

Følgende algoritme repræsenterer skolealgoritmen til multiplikation af heltal, jfr. også [Algoritmisk Problemløsning] afsnit 1.6.

**Algoritme:** Multiplikation
**Stimulans:** $x, y \geq 0$
**Respons:** $z = x * y$
**Metode:** $z, i := 0, 0$
  **do** $\{I\}$
    $i < ||y|| \rightarrow$
      $z := z + ((x \otimes y_i) \uparrow i)$
      $i := i + 1$
  **od**

Her angiver $||y||$ antallet af cifre i $y$, $y_i$ angiver det $i$'te ciffer i $y$, $\otimes$ betyder multiplikation af et tal med et enkelt ciffer, og $h \uparrow i$ betyder $h * 10^i$.

> Angiv en gyldig invariant $I$ for algoritmen, som er nyttig i forbindelse med et korrekthedsbevis (det kræves ikke, at beviset gennemføres).

Algoritmen Multiplikation kan omskrives til en procedure på følgende måde

```
Proc Mult(x, y: Int) → (Int)
   (+ Var z, i: Int
      z, i := 0, 0
      do {I}
         i < ||y|| →
            z := z + ((x ⊗ y_i) ↑ i)
            i := i + 1
      od
      return z
   +)
end Mult
```

Angiv en specifikation af Mult. Bevis dernæst, at nedenstående algoritme er gyldig og korrekt.

**Algoritme:** Itereret kvadrering
**Stimulans:** $x, n \geq 0$
**Respons:** $r = x^{2^n}$
**Metode:** $r, i := x, 0$

$\quad$ **do** $\{(r = x^{2^i}) \wedge (0 \leq i \leq n)\}$
$\quad\quad$ $i < n \rightarrow$
$\quad\quad\quad$ $r :=$ Mult($r$,$r$)
$\quad\quad\quad$ $i := i + 1$
$\quad$ **od**

## Opgave 3 (10%)

En tekst er som bekendt et ***palindrom***, hvis den er lig med sin egen spejling, det vil sige, hvis den opfylder prædikatet

$$\boldsymbol{pal}(t): \ (\forall i \in 0..|t|: \ t.(i) = t.(|t|-i-1))$$

Man kan nemt afgøre om $t$ er et palindrom i tid $O(|t|)$.

| |
|---|
| Angiv en algoritme $k$-pal, der afgør om en tekst $t$ har en deltekst af længde $k$, der er et palindrom. Angiv tidskompleksiteten $T[\![k\text{-pal}]\!](|t|, k)$. |

| |
|---|
| Angiv også en algoritme, der afgør om $t$ har en deltekst af en eller anden længde $\geq 2$, der er et palindrom. Hvad er den bedste tidskompleksitet, du kan opnå? |

## Opgave 4 (10%)

Betragt følgende TRINE program.

```
Process P
    (+ Type A = Prod(x: Int, y: B)
       Type B = List(C)
       Type C = Prod(x: D, y: List(A))
       Type D = Int
       Type E = List(D)
       Type F = List(List(Int))

       Var a: A
       Var c: C
       Var e: E
       Var f: F

       a := Prod(7, List())
       c := a
       e := List()
       f := e
    +)
end P
```

| Vil det blive accepteret af TRINE oversætteren? Begrund dit svar. |
|---|

## Opgave 5 (15%)

Der skal konstrueres en box HR (for Højde Register) med følgende udseende

> **Box** HR
>     **Type** R = ≪mængde af par af formen (p,h)≫
>     **Proc** Init [r: R]
>     **Proc** Insert [r: R] (p, h: Int)
>     **Proc** Delete [r: R] (p: Int)
>     **Proc** Member [r: R] (p: Int) → (Bool)
>     **Proc** Height [r: R] (p: Int) → (Int)
>     **Proc** Who [r: R] (h: Int) → (Vector)
> **end** HR

som realiserer en datastruktur, der indeholder information om et antal personer (identificeret ved deres personnummer p) og deres højde h (angivet i centimeter). Procedurerne Init, Insert, Delete og Member virker som sædvanligt, medens Height [r] (p) skal returnere højden på personen p som angivet i registeret r, og Who [r] (h) skal returnere en liste med numre på de personer, der i registeret r har højden h.

| |
|---|
| Angiv en specifikation af proceduren Height. |

| |
|---|
| Beskriv en realisation af typen R så Init får tidskompleksitet $O(1)$, så Insert, Delete, Member og Height får tidskompleksitet $O(\log \lvert r \rvert)$, og så Who får tidskompleksitet $O(\lvert w \rvert)$, hvor $w$ er den returnerede liste. |

## Opgave 6 (15%)

Antag, at der skal udlægges et kommunikationsnetværk mellem byer i Jylland, således at alle byer er indbyrdes forbundne (eventuelt via andre byer) og den samlede længde af kablerne bliver så kort som mulig.

> Hvordan kan man løse dette problem? Angiv algoritmens tidskompleksitet i den generelle situation, hvor der er $n$ byer, der skal forbindes.

Kommunikationslinjer kan blive afbrudt og kan blive repareret igen. Når en linje bliver afbrudt vil der normalt være byer, der ikke længere kan komme i forbindelse med hinanden; når der sker reparationer kan hidtil brudte forbindelser blive genetableret. Man er derfor interesseret i til stadighed at vedligeholde oplysninger om, hvilke byer der er forbindelser imellem.

> Angiv en passende repræsentation af en sådan "forbindelses-information", og angiv ligeledes hvilke algoritmer, der skal benyttes ved afbrydelse og genetablering af kommunikationslinjerne. Angiv algoritmernes tidskompleksiteter.

## Opgave 7 (10%)

Betragt talrækken $(A_n)_{n\geq 0} = (1, 1, 3, 4, 8, 11, 21, 29, 55, \ldots)$ defineret ved:

$$
\begin{array}{ll}
A_1 = A_2 = 1 & \\
A_n = B_{n-1} + A_{n-2} & n > 2 \\
B_1 = B_2 = 2 & \\
B_n = A_{n-1} + B_{n-2} & n > 2
\end{array}
$$

der beregnes af fx følgende TRINE værdiprocedure A (der er gensidigt rekursiv med værdiproceduren B):

```
Proc A(n: Int) → (Int)
    if n<1 → abort
    ff n ≤ 2 → return 1
    ff true → return B(n-1)+A(n-2)
    fi
end A

Proc B(n: Int) → (Int)
    if n<1 → abort
    ff n ≤ 2 → return 2
    ff true → return A(n-1)+B(n-2)
    fi
end B
```

| |
|---|
| Skriv en modificeret udgave af A og B, der benytter dynamisk programmering til at undgå en eksponentiel tidskompleksitet. Angiv den forbedrede tidskompleksitet. |

## Opgave 8 (10%)

Betragt følgende to eksempler på realiseringer af del-og-kombiner skabelonen. Procedurerne gør ikke noget fornuftigt; de tjener kun til at illustrere forskellige tidskompleksiteter.

```
Proc X[t: Text]
  if | t |>1 →
      (+ Var t1, t2: Text
        t1, t2 := t(0 .. | t |/2), t(| t |/2 .. | t |)
        X[t1]
        X[t2]
        (+ Var i, j: Int
          i := 0
          do i<| t | →
              j := 0
              do j<| t | → j := j+1 od
              i := i+1
          od
        +)
      +)
  fi
end X

Proc Y(n: Int)
  if n>1 →
      (+ Var i, j: Int
        i, j := 0, 0
        do i*i<n → i, j := j, j+1 od
      +)
      Y(n/3)
      Y(n/3)
  fi
end Y
```

| |
|---|
| Angiv deres tidskompleksiteter. Begrund dit svar. |