Skriftlig eksamen
Datastrukturer og algoritmer

Vinter 2000/01

Opgavesættet består af 4 opgaver, der ved bedømmelsen tillægges følgende vægte:

Opgave 1	20%
Opgave 2	20%
Opgave 3	30%
Opgave 4	30%

Alle sædvanlige hjælpemidler er tilladt. I tilfælde af unøjagtigheder i opgaveteksterne forventes det, at deltagerne selv præciserer besvarelsernes forudsætninger.

Opgavesættet består af 5 paginerede sider (incl. forsiden). Kontroller at din kopi er fuldstændig.

Opgave 1: Binære træer (20%)

Lad et binært træ være repræsenteret ved hjælp af nedenstående klasse:

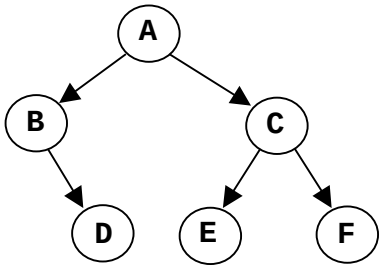
```
class Node {
    Node left, right;
    char key;

    Node(char k, Node l, Node r) {
        key = k; left = l; right = r;
    }
}
```

idet ethvert tomt træ refereres med referencekonstanten `null`.

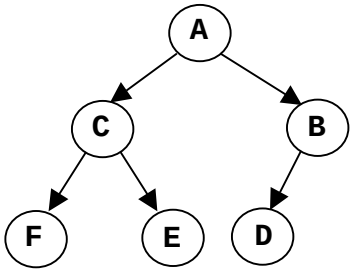
Spørgsmål 1.1 Programmér en metode

```
void reflect(Node t)
```

der, givet roden `t` i et binært træ, spejlvender det pågældende træ. Således skal træet



omdannes til



Spørgsmål 1.2 Programmér en metode

```
Node reflection(Node t)
```

der, givet roden `t` i et binært træ, returnerer et helt nyt træ, som er spejlbilledet af det givne træ.

Opgave 2: Søgning (20%)

En sekvens `a[1]`, `a[2]`, ..., `a[n]` af reelle tal siges at være *cyklisk sorteret*, hvis der findes et indeks `i`, således at sekvensen

`a[i]`, `a[i+1]`, ..., `a[n]`, `a[1]`, ..., `a[i-1]`

er sorteret i stigende orden.

Vi ønsker at udvikle en algoritme, der kan finde indekset på det mindste element i en cyklisk sorteret sekvens. For at forenkle algoritmen antages, at dette indeks er unikt.

Det ønskede indeks kan let bestemmes ved et fuldstændigt gennemløb af sekvensen.

Spørgsmål 2.1 Programmér en metode

```
int minIndex(double[] a, int n)
```

der realiserer denne algoritme. Bemærk at elementet `a[0]` ikke benyttes.

Ovennævnte algoritme udnytter imidlertid ikke, at sekvensen er cyklisk sorteret. En mere effektiv algoritme kan opnås ved at benytte del-og-hersk-princippet. Her benyttes ideen fra binær søgning til successivt at halvere det indeksinterval, der søges i.

Spørgsmål 2.2 Programmér en udgave af metoden `minIndex`, der benytter denne ide.

Opgave 3: Sortering (30%)

Nedenfor ses en algoritme til sortering.

```
int i = 2;
while (i <= n) {
    if (i > 1 && a[i] < a[i - 1]) {
        swap(a, i, i - 1);
        i = i - 1;
    }
    else
        i = i + 1;
}
```

Spørgsmål 3.1 Metoden `swap` benyttes til at foretage parvise ombytninger af elementerne i arrayet `a`. Programmér denne metode, idet elementerne i `a` antages at være af typen `int`.

Spørgsmål 3.2 Argumenter for at, hvis algoritmen terminerer, så vil elementerne a[1], a[2], ... , a[n] efter udførelse være blevet sorteret i stigende orden. Der kræves ikke et formelt bevis.
Vink: Find en passende løkkeinvariant (et udsagn, der altid er sandt før løkketesten `i <= n`).

Spørgsmål 3.3 Argumenter for at algoritmen terminerer. Der kræves ikke et formelt bevis.

Spørgsmål 3.4 Angiv algoritmens tidskompleksitet med O-notation.

Opgave 4. Rød-sort-træer (30%)

Lad et rød-sort-træ være repræsenteret ved objekter af følgende to klasser:

```
public class RedBlackTree() {
    public void insert(Comparable x) { /* kode udeladt */ }
    public Comparable find(Comparable x) { /* kode udeladt */ }
    public void remove(Comparable x) { /* kode udeladt */ }
    public boolean isEmpty() { return root == null; }
    public void check() { /* se spørgsmål 4.1 */ }

    BinaryNode root;
    static final int RED = 0, BLACK = 1;
}

class BinaryNode() {
    BinaryNode left, right;
    int color;
    Comparable element;
}
```

Et `BinaryNode`-objekt repræsenterer en knude i det binære træ, idet `left` og `right` refererer til henholdsvis roden i knudens venstre og højre undertræ (`null`, hvis undertræet er tomt). Feltet `color` angiver knudens aktuelle farve (`RED` eller `BLACK`).

Elementerne i træets knuder er objekter af en klasse, der implementerer følgende interface:

```
public interface Comparable {
    int compares(Comparable rhs);
}
```

Sammenligninger foretages ved hjælp af metoden `compares`. Kaldet `lhs.compares(rhs)`, hvor `lhs` og `rhs` er to `Comparable`-objekter, returnerer `-1`, `0` eller `1`, alt efter om `lhs` er mindre end, lig med, eller større end `rhs`.

Til testformål ønskes nu implementeret en metode `check` i klassen `RedBlackTree`, som kan benyttes til at undersøge om et givet træ faktisk er et rød-sort-træ, d.v.s. om det opfylder følgende betingelser:

(1) Træet er et søgetræ
(2) Roden er sort
(3) Der er ikke to konsekutive røde knuder på nogen vej fra rod til blad
(4) Enhver vej fra rod til blad indeholder det samme antal sorte knuder

Metoden skal udskrive en fejlmeddelelse, hvis blot én af betingelserne ikke er opfyldt.

Spørgsmål 4.1 Programmér metoden `check`. **Vink**. Benyt en eller flere hjælpemetoder.