Opgaveløsninger
(sæt 5)

Opgave 16: 7.19 (1 point)

Metoden nedenfor forudsætter, at n er ikke-negativ.

```
int numberOfOnes(int n) {
    return n > 0 ? numberOfOnes(n / 2) + n % 2 : 0;
}
```

Eftertanke: Hvis metoden omskrives til

```
int numberOfOnes(int n) {
    return n > 0 ? n % 2 + numberOfOnes(n / 2) : 0;
}
```

er der tale om *halerekursion*. Den kan derfor let omskrives til en iterativ metode:

```
int numberOfOnes(int n) {
    int numberOfOnes = 0;
    for ( ; n > 0; n /= 2)
        numberOfOnes += n % 2;
    return numberOfOnes;
}
```

Opgave 17: 7.20 (2 point)

```
public int binarySearch(Comparable[] a, Comparable x) {
   return binarySearch(a, x, 0, a.length - 1);
}

private int binarySearch(Comparable[] a, Comparable x,
                         int low, int high) {
    if (low == high)
        return a[low].compareTo(x) == 0 ? low : -1;
    int mid = (low + high)/2;
    return a[mid].compareTo(x) < 0 ?
           binarySearch(a, x, mid + 1, high) :
           binarySearch(a, x, low, mid);
}
```

Opgave 18: 7.25 (3 point, ikke-obligatorisk)

```
public void permute(String str) {
    permute(str.toCharArray(), 0, str.length() - 1);
}

private void permute(char[] str, int low, int high) {
    if (low > high)
        System.out.println(str);
    for (int i = low; i <= high; i++) {
        swap(str, low, i);
        permute(str, low + 1, high);
        swap(str, low, i);
    }
}

private void swap(char[] s, int i, int j) {
    char t = s[i]; s[i] = s[j]; s[j] = t;
}
```

Kaldet `permute(str, low, high)` skal udskrive alle permutationer af `str[low:high]`. Efter kaldet skal `str[low:high]` være uændret.

En udgave, der benytter sig af kloning, er vist nedenfor:

```
private void permute(char[] str, int low, int high) {
    if (low > high)
        System.out.println(str);
    for (int i = low; i <= high; i++) {
        char[] tmp = str.clone();
        tmp[i] = str[low];
        tmp[low] = str[i];
        permute(tmp, low + 1, high);
    }
}
```

Opgave 19: N-dronning problemet (4 point, ikke-obligatorisk)

(a)

Nedenstående program løser 8-dronning problemet ved hjælp af baksporing.

Metoden `placeQueens` placerer systematisk alle resterende dronninger på en given række og alle efterfølgende rækker, således at to dronninger på brættet ikke kan slå hinanden.

Til at afgøre om der står en dronning i en given søjle benyttes heltalsarrayet `q`. Værdien af `q[col]` er `0`, hvis og kun hvis søjlen `col` er fri. Hvis søjlen ikke er fri, er værdien lig med rækkenummeret på den dronning, der står i søjlen.

Til at afgøre, om der står en dronning på en given diagonal benyttes de to booleske arrays `up` og `down`. `up[row+col-2]` er `false`, hvis og kun op-diagonalen igennem feltet (`row`,`col`) er fri. Tilsvarende er `down[col-row+7]` `false`, hvis og kun ned-diagonalen igennem feltet (`row`,`col`) er fri. Her er udnyttet, at for en op-diagonal er summen af rækkenummer og søjlenummer for ethvert felt på diagonalen konstant, og for en ned-diagonal, at differensen af rækkenummer og søjlenummer for ethvert felt på diagonalen er konstant.

```
public class EightQueenProblem {
    static int[] q = new int[9];
    static boolean[] down = new boolean[15], up = new boolean[15];

    static void placeQueens(int row) {
        for (int col = 1; col <= 8; col++) {
            if (q[col] == 0 && !up[row+col-2] && !down[col-row+7]) {
                q[col] = row; up[row+col-2] = down[col-row+7] = true;
                if (row == 8) {
                    for (int i = 1; i <= 8; i++)
                        System.out.print("(" + q[i] + "," + i + ") ");
                    System.out.println();
                }
                placeQueens(row + 1);
                q[col] = 0; up[row+col-2] = down[col-row+7] = false;
            }
        }
    }

    public static void main(String args[]) {
        placeQueens(1);
    }
}
```

(b)

```
public class NQueenProblem {
    public NQueenProblem(int n) {
        this.n = n;
        q = new int[n + 1];
        down = new boolean[2*n - 1];
        up = new boolean[2*n - 1];
    }

    final int n;
    int[] q;
    boolean[] down, up;
    int solutions;

    public int solutions() {
        placeQueens(1);
        return solutions;
    }

    void placeQueens(int row) {
        for (int col = 1; col <= n; col++) {
            if (q[col] == 0 && !up[row+col-2] && !down[col-row+n-1]) {
                if (row == n)
                    solutions++;
                else {
                    q[col] = row; up[row+col-2] = down[col-row+n-1] = true;
                    placeQueens(row + 1);
                    q[col] = 0; up[row+col-2] = down[col-row+n-1] = false;
                }
            }
        }
    }

    public static void main(String args[]) {
        for (int n = 4; n <= 12; n++) {
            long t = System.currentTimeMillis();
            System.out.println("n = " + n + ", solutions = " +
                   new NQueenProblem(n).solutions() + ", time = " +
                   (System.currentTimeMillis() - t) / 1000.0 + " seconds");
        }
    }
}
```

En kørsel af programmet på en 400 MHz Macintosh PowerBook G3 gav følgende udskrift:

```
n = 4, solutions = 2, time = 0.011 seconds
n = 5, solutions = 10, time = 0.0 seconds
n = 6, solutions = 4, time = 0.0 seconds
n = 7, solutions = 40, time = 0.0010 seconds
n = 8, solutions = 92, time = 0.0010 seconds
n = 9, solutions = 352, time = 0.0050 seconds
n = 10, solutions = 724, time = 0.022 seconds
n = 11, solutions = 2680, time = 0.181 seconds
n = 12, solutions = 14200, time = 1.499 seconds
```