Opgaveløsninger (sæt 4)

Opgave 1a (8.1)

For at forenkle besvarelsen af opgaven indføres en metode, `comparator`, der sammenligner to elementer i et array og ombytter dem, hvis det første er større end det andet.

```
void comparator(int i, int j) {
    if (a[i] > a[j])
        swap(a, i, j);
}
```

En mulig løsning af opgaven er da følgende:

```
comparator(1,2);
comparator(2,3);
comparator(3,4);  // nu er a[4] maksimum af de 4 elementer
comparator(1,2);
compatator(2,3);  // nu er a[3] maksimum af de restende 3
compatator(1,2);  // nu er a[2] maksimum af de restende 2,
                  // og dermed er a[1] det mindste af de 4 elementer
```

Imidlertid kan man ved at bruge de små grå finde frem til en løsning, der bruger en sammenligning mindre.

```
comparator(1,2);
comparator(3,4);
comparator(1,3);  // nu er a[1] minimum af de 4 elementer
comparator(2,4);  // nu er a[4] maksimum af de 4 elementer
comparator(2,3);  // nu er a[2] og a[3] ordnet,
                  // og dermed er alle 4 elementer sorteret
```

Opgave 1b (8.2)

Hvis dataene er sorteret i forvejen, udfører ingen af de tre metoder ombytninger eller flytninger af elementer, men de foretager sammenligninger. Antallet af sammenligninger er:

for `selection`: (n-1)+(n-2)+...+1 = n(n-1)/2, altså $O(n^2)$
for `insertion`: n-1, altså O(n)
for `bubble`: (n-1)+(n-2)+...+1 = n(n-1)/2, altså $O(n^2)$

Dvs. `insertion` er hurtigst i dette tilfælde.

Opgave 1c (8.3)

Hvis dataene er i omvendt orden, kommer antallet af ombytninger og flytninger til at spille en rolle. Effektiviteten opgøres efter antallet af sammenligninger, antal ombytninger og antal flytninger.

Antal sammenligninger:

for `selection`: (n-1)+(n-2)+...+1 = n(n-1)/2, altså $O(n^2)$
for `insertion`: (n-1)+(n-2)+...+1 = n(n-1)/2, altså $O(n^2)$
for `bubble`: (n-1)+(n-2)+...+1 = n(n-1)/2, altså $O(n^2)$

Antal ombytninger:

for `selection`: cirka n/2 , altså O(n)
for `insertion`: ingen
for `bubble`: (n-1)+(n-2)+...+1 = n(n-1)/2, altså $O(n^2)$

Antal flytninger:

for `selection`: ingen
for `insertion`: (n-1)+(n-2)+...+1 = n(n-1)/2, altså $O(n^2)$
for `bubble`: ingen

Idet en ombytning i praksis kun er er lille faktor (f.eks. 3) dyrere end en flytning, er det oplagt, at `selection` vil være den hurtigste (når 3(n-1) < n(n-1)/2, dvs. når n > 6).

Opgave 1d (8.8)

Betragt filen $B_1\ B_2\ A_3$, hvor et bogstav udgør nøglen, mens indeks angiver postens oprindelige placering i filen. Efter sortering med `selection`, bliver resultatet $A_3\ B_2\ B_1$ (idet A_3 bytter plads med B_1 i første iteration af den yderste løkke). Metoden er derfor ikke stabil.

For de to andre metoder sker der derimod aldrig ombytninger, når to elementer er lige store, og det må derfor konkluderes, at de er stabile.

Opgave 2

Nedenstående løsning benytter sig af boblesortering.

```
public void sort() {
    if (first == null)
        return;
    boolean done = false;
    while (!done) {
        done = true;
        Node prev = null, n = first;
        while (n.next != null) {
           if (n.key > n.next.key) {
               done = false;
               Node t = n.next;
               if (prev == null)
                   first = t;
               else
                   prev.next = t;
               n.next = t.next;
               t.next = n;
               prev = n;
               n = t;
            }
            else {
                prev = n;
                n = n.next;
            }
        }
    }
}
```