# SI-möte #9, Programkonstruktion och datastrukturer
*Lösningsförslag*

Elias Castegren
elca7381@student.uu.se

1 februari 2011

## Övningar

**1.** –

**2.**

| | Best | Average | Worst |
|---|---|---|---|
| Insertion sort | $\Theta(n)$ | $\Theta(n^2)$ | $\Theta(n^2)$ |
| Merge sort | $\Theta(n \log n)$ | $\Theta(n \log n)$ | $\Theta(n \log n)$ |
| Quick sort | $\Theta(n \log n)$ | $\Theta(n \log n)$ | $\Theta(n^2)$ |

**3.**
Valet av sorteringsalgoritm beror på vilken datastruktur som ska sorteras, storleken på datamängden, om man är beroende av stabil sortering eller inte, och så vidare. Insertion sort kan fungera bra om man till exempel vet att man bara kommer sortera väldigt små datamängder.

**4.**
Se föreläsningsslides.

**5.**
Man kan gå igenom listan en gång och lägga vartannat element i den ena listan och vartannat i den andra:

```
fun split'([], _ (l1, l2)) = (l1, l2)
  | split'(f::r, left, (l1, l2)) =
        split'(r, not left, if even then (f::l1, l2) else (l1, f::l2);

fun split(l) = split'(l, true, ([], []));
```

Sortering med den nya delningsfunktionen är dock inte längre stabil (prova att sortera `[(1, "a''), (1, "b''), (1, "c''), (1, "d'')]` efter heltalsnyckeln).

Eftersom komplexiteten för delningsfunktionen fortfarande är $\Theta(n)$ så påverkas inte heller komplexiteten för merge sort.

**6.**
Se föreläsningsslides.

**7.**
För att göra sorteringsalgoritmerna polymorfa (alltså tillåta sortering på nycklar som har en godtycklig typ) måste man skicka med en jämförelsefunktion som argument och använda den istället för den vanliga "mindre än"-operatorn. Anropen skulle kunna se ut så här:

```
sort(["bertil", "adam"], String.<) = ["adam", "bertil"]
sort([3.14159, 2.71828], Real.<) = [2.71828, 3.14159]
sort([42, 13], Int.<) = [13, 42]
```

**8.**
Säg att man har en representation av en vanlig kortlek som också tillåter att man skriver något på varje kort. Anta att man har alla korten i oordning i en lista och nu vill sortera korten efter färg och valör, till exempel alla hjärter, följt av alla spader, sen ruter och sist klöver. Eftersom man vet från början att det finns exakt 52 kort kan man till exempel lagra dem i en vektor med 52 platser och lägga in varje kort direkt på rätt plats. Funktionen skulle kunna se ut ungefär så här:

```
fun cardListToVector([], V) = V
  | cardListToVector(f::r, V) =
        let
            val c = case getColor(f) of
                        hearts => 0
                      | spades => 1
                      | diamonds => 2
                      | _ => 3
            val v = getValue(f)
        in
            cardListToVector(r, Vector.update(V, c*13 + v - 1, f))
        end;
```

där `cardListToVector` anropas med kortlistan och en vektor (av typen `card vector`) med 52 element.