SI-möte #13, Programkonstruktion och datastrukturer

Elias Castegren
elca7381@student.uu.se

Övningar

1.
Betrakta nedanstående kod:

```
fun reverse [] = []
  | reverse (f::r) = reverse(r) @ [f];
```

Beräkna funktionens tidskomplexitet. Skriv sedan en svansrekursiv version, utan användning av append. Beräkna tidskomplexiteten för den nya versionen av `reverse`.

2.
Följande funktion beräknar produkten av två heltal a och b:

```
fun russian(1,b) = b
  | russian(a,b) =
        if a mod 2 = 1 then
            b + russian(a div 2, b*2)
        else
            russian(a div 2, b*2);
```

Vad har funktionen för tidskomplexitet? Skriv en svansrekursiv version av samma funktion. Vad har den nya funktionen för komplexitet?

3.
Beräkna nedanstående funktions tidskomplexitet. Skriv en svansrekursiv version av samma funktion. Är tidskomplexiteten densamma? Är resultatet detsamma?

```
fun killDuplicates([]) = []
  | killDuplicates(f::r) =
    let
        fun kill(_, []) = []
          | kill(e, f::r) = if e=f then kill(e, r) else f::kill(e, r);
    in
        f::killDuplicates(kill(f,r))
    end;
```

4.
Beräkna tidskomplexiteten för nedanstående två funktioner som använder *balanserade* binära sökträd:

```
fun dataFromKey(_, Leaf) = NONE
  | dataFromKey(k, Node(k', d, L, R)) =
        if k>k' then
            dataFromKey(k, R)
        else if k<k' then
           dataFromKey(k, L)
        else
            SOME d;

fun keyFromData(d, Leaf) = NONE
  | keyFromData(d, Node(k, d', L, R)) =
        if d = d' then
            SOME k
        else
            let
                val k' = keyFromData(d, L)
            in
                if isSome k' then
                    k'
                else
                    keyFromData(d, R)
            end;
```

5.
Vad är `kalimba`s tidskomplexitet?

```
fun kalimba(v) =
    let
        val i = ref 0
        val j = ref 0
    in
        while !i < Vector.length(!v) do
            (j := !i + 1;
             while !j < Vector.length(!v) do
                 (if Vector.sub(!v,!j) = Vector.sub(!v,!i) then
                      v := Vector.update(!v, !j, Vector.sub(!v, !j)+1)
                  else
                      ();
                  j := !j + 1);
             i := !i + 1);
   end;
```

Vad blir skillnaden om `Vector.length` har linjär eller konstant komplexitet?