5.3

Strukturel induktion.

Mængden S er defineret rekursivt ved:
Basisskridt: vi har angivet et eller flere elementer, der tilhører S
Rekursionsskridt: vi har angivet en eller flere regler, der hver ud fra et eller flere elementer i S konstruerer et nyt element i S.

Lad $P(x)$ være et åbent udsagn, $x \in S$.

For at bevise at $P(x)$ er sand for alle $x \in S$ skal vi:
Basisskridt: bevise at $P(x)$ er sand for ethvert x indført i basisskridtet af definitionen af S
Rekursionsskridt: bevise at hvis x er konstrueret fra $x_1, \ldots, x_\ell$ i rekursionsskridtet af definitionen af S og hvis $P(x_1), \ldots, P(x_\ell)$ er sande så er $P(x)$ sand.

Strenge

Σ : et alfabet, altså en endelig mængde af symboler.

Definition. Σ^*, mængden af strenge over Σ defineres ved:
Basisskridt: Den tomme streng $\lambda \in \Sigma^*$.
Rekursionsskridt: Hvis $w \in \Sigma^*$ og $x \in \Sigma$ så er $wx \in \Sigma^*$.

Vise $P(x)$ Sand for alle $x \in S$

Basisskridt: $P(\lambda)$ Sand

Rekursionsskridt: Hvis $P(w)$ Sand så er $P(wx)$ Sand.

Definition. Konkatenering af strenge $w_1 \cdot w_2$ af to strenge $w_1, w_2 \in \Sigma^*$ defineres ved:

Basisskridt: $w_1 \cdot \lambda = w_1$

Rekursionsskridt: Hvis $w_2 \in \Sigma^*$ så er

$$w_1 \cdot (w_2 x) = (w_1 \cdot w_2) x.$$

Definition. Længden $\ell(w)$ af en streng w defineres ved

Basisskridt: $\ell(\lambda) = 0$

Rekursionsskridt: Hvis $w = w_1 x$, hvor $w_1 \in \Sigma^*$ og $x \in \Sigma$ så er $\ell(w) = \ell(w_1) + 1$.

Sætning (Eks 12). $\ell(w_1 \cdot w_2) = \ell(w_1) + \ell(w_2)$.

Bevis

Valg fast w_1 og brug strukturel induktion efter w_2.

Basisskridt $(w_2 = \lambda)$: $\ell(w_1 \cdot w_2) = \ell(w_1 \cdot \lambda) =$

$\ell(w_1) = \ell(w_1) + 0 = \ell(w_1) + \ell(\lambda) = \ell(w_1) + \ell(w_2)$

Rekursionsskridt Lad $w \in \Sigma^*$ og antag

$$l(w_1 \cdot w) = l(w_1) + l(w).$$

Lad $w_2 = wx$ hvor $x \in \Sigma$

$$l(w_1 \cdot w_2) = l(w_1 \cdot (wx)) = l((w_1 \cdot w)x) =$$

$$l(w_1 \cdot w) + 1 = l(w_1) + l(w) + 1 =$$

↑ antagelse

$$l(w_1) + l(wx) = l(w_1) + l(w_2).$$

Sætningen gælder dermed for alle $w_1, w_2 \in \Sigma^*$

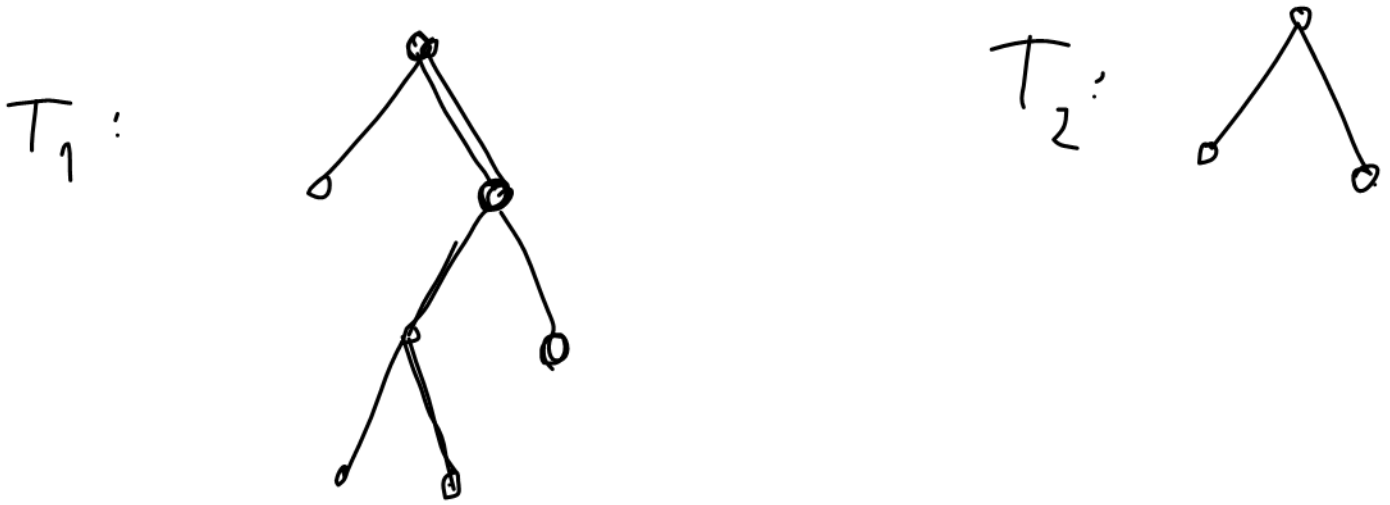
□

Binære træer

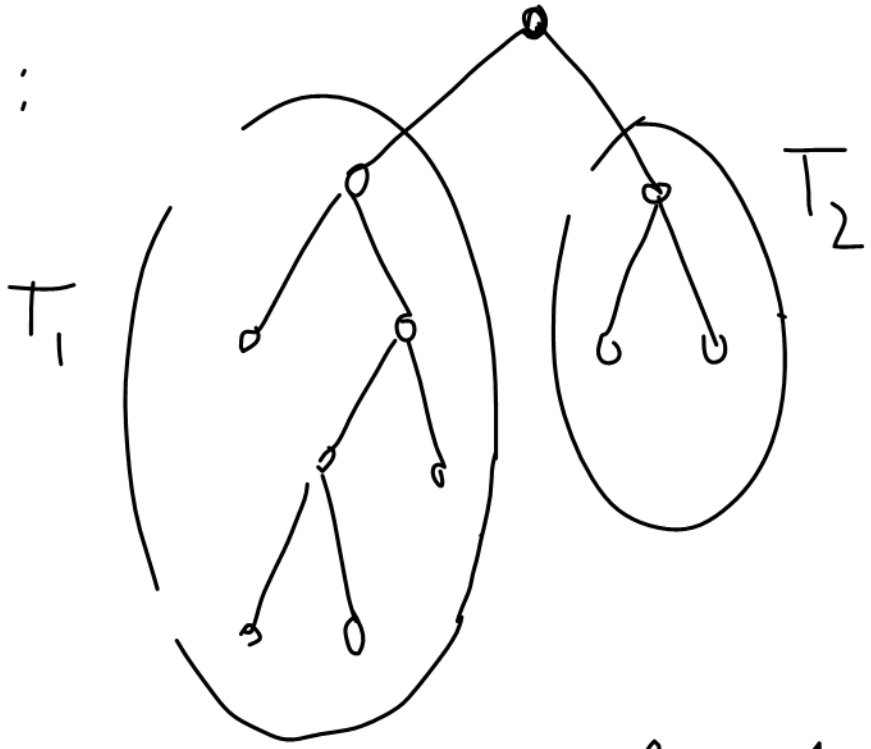
Definition. Mængden af fulde binære træer kan defineres ved:

Basisskridt: Et træ, der består ét punkt r, er et fuldt binært træ med rod r.

Rekursionsskridt: Hvis T_1 og T_2 er fulde binære træer så er $T_1 \cdot T_2$ et fuldt binært træ, der består af T_1, T_2 og en rod r samt en kant fra r til roden af T_1 og en kant fra r til roden af T_2.



$T_1 \cdot T_2$:



Vise $P(T)$ er sand for alle trær T som fyller kravet til et binært tre.

<u>Basissteg</u> Hvis $T = r$ så er $P(T)$ sann.

<u>Rekursjonssteg</u> Hvis $P(T_1)$, $P(T_2)$ sanne så er $P(T_1 \cdot T_2)$ sann.

$n(T)$: antal punkter i et fuldt binært træ T opfylder:
Basisskridt: Hvis T består af en rod så er $n(T) = 1$.
Rekursionsskridt: Hvis $T = T_1 \cdot T_2$ så er

$$n(T) = n(T_1) + n(T_2) + 1.$$

$h(T)$: højden af et fuldt binært træ T defines ved
Basisskridt: Hvis T består af en rod så er $h(T) = 0$.
Rekursionsskridt: Hvis $T = T_1 \cdot T_2$ så er

$$h(T) = \max\{h(T_1), h(T_2)\} + 1.$$

Sætning. Et fuldt binært træ T opfylder

$$n(T) \leq 2^{h(T)+1} - 1.$$

Bevis ved strukturel induktion

Basisskridt: $(T = r)$ $\quad n(T) = 1$, $h(T) = 0$

$$2^{h(T)+1} - 1 = 2^{0+1} - 1 = 2 - 1 = 1$$

Rekursionsskridt: Lad T_1 og T_2 være fulde binære træer. Og antag

$$n(T_1) \leq 2^{h(T_1)+1} - 1, \quad n(T_2) \leq 2^{h(T_2)+1} - 1$$

Sæt $T = T_1 \cdot T_2$

$$n(T) = n(T_1) + n(T_2) + 1 \leq \quad \text{(antagelse)}$$

$$2^{h(T_1)+1} - 1 + 2^{h(T_2)+1} - 1 + 1 =$$

$$2 \cdot \left(2^{h(T_1)} + 2^{h(T_2)} \right) - 1.$$

$$h(T) = \max \{ h(T_1), h(T_2) \} + 1$$

$$m = \max \{ h(T_1), h(T_2) \}, \quad h(T) = m + 1$$

$$2 \left(2^{h(T_1)} + 2^{h(T_2)} \right) - 1 \leq 2 \cdot \left(2^m + 2^m \right) - 1$$

$$= 2 \cdot 2 \cdot 2^m - 1 = 2^{m+2} - 1 = 2^{h(T)+1} - 1$$

□

5.4

EKS $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$

Rekursiv definition

Basis: $0! = 1$

Rekursion: $n! = n \cdot (n-1)!$ for $n \geq 1$

Algoritme: Iterativ beregning af $n!$

procedure iterativ factorial(n: ikke-negativt heltal)
$fac := 1$
for $i := 1$ **to** n
    $fac := i \cdot fac$
return fac
$\{fac = n!\}$

Algoritme 1: Rekursiv beregning af $n!$ Side 354

procedure *factorial*(n: ikke-negativt heltal)
if $n = 0$ **then return** 1
else return $n \cdot$ *factorial*$(n - 1)$
{outputtet er $n!$}

Algoritme 3:

Rekursiv beregning af største fælles divisor

procedure gcd(a, b: heltal, hvor $0 \leq a < b$)
if $a = 0$ **then return** b
else return gcd($b \mod a, a$)
{outputtet er gcd(a, b)}

Lemma i afsnit 4.3:

$$\gcd(a, b) = \gcd(b \bmod a, a)$$

Fibonnacci tal definuet rekursivt

$$f_0 = 0$$

$$f_1 = 1$$

$$f_n = f_{n-1} + f_{n-2} \qquad n \geq 2$$

$f_2 = f_1 + f_0 = 1$, $f_3 = f_2 + f_1 = 2$, $f_4 = f_3 + f_2 = 3$

$f_5 = f_4 + f_3 = 5$

Algoritme 8:

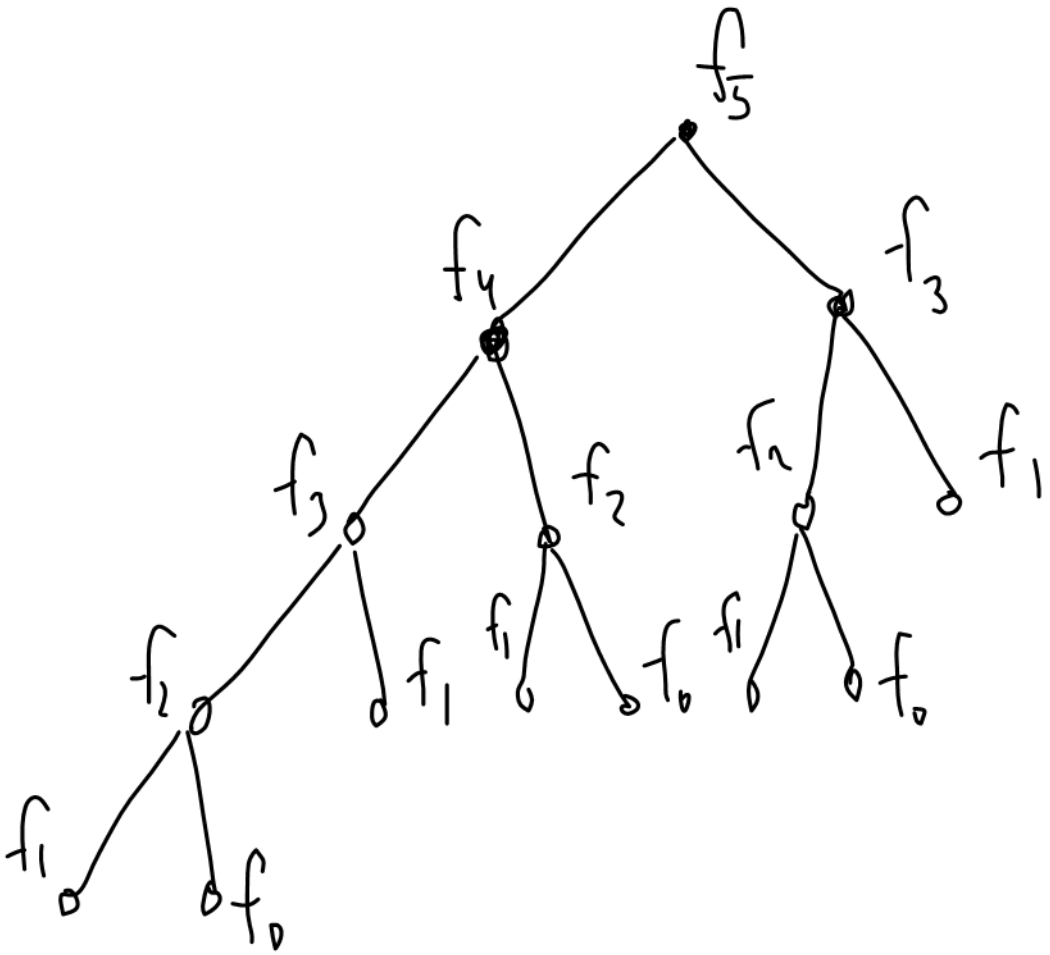
Iterativ beregning af Fibonaccital

procedure iterativ fibonacci (n: ikke-negativt heltal)
if $n = 0$ **then return** 0
else
    $x := 0$
    $y := 1$
    for $i := 1$ **to** $n - 1$
        $z := x + y$
        $x := y$
        $y := z$
    return y
{y er det n'te Fibonacci tal.}

Algoritme 7:

Rekursiv beregning af Fibonaccital

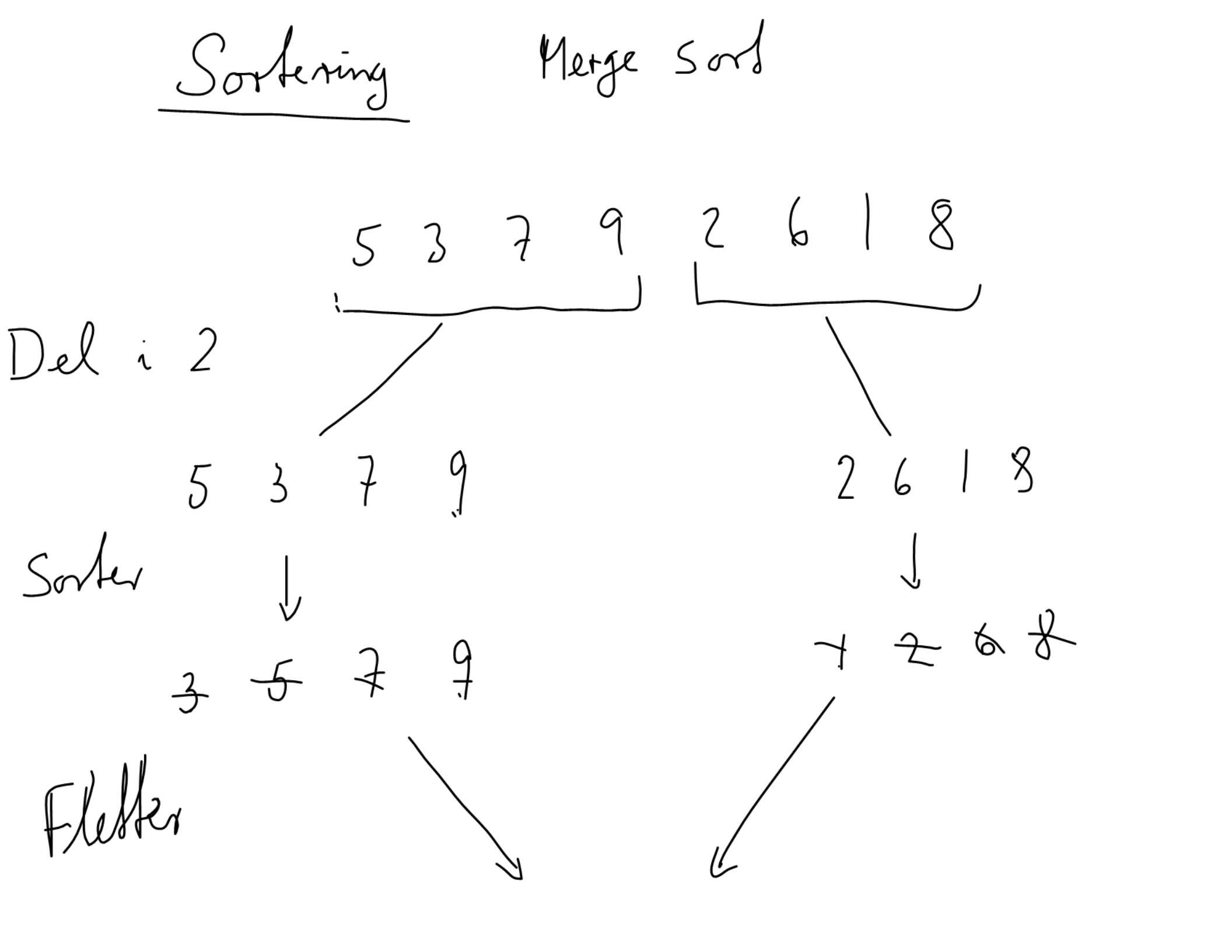
procedure fibonacci (n: ikke-negativt heltal)
if $n = 0$ **then** ~~fibonacci(0) := 0~~ return 0
else
    if $n = 1$ **then** ~~fibonacci(1) := 1~~ return 1
    else return fibonacci$(n-1)$ + fibonacci$(n-2)$
{outputtet er det n'te Fibonacci tal.}


f_5
f_4
f_3
f_3
f_2
f_2
f_1
f_2
f_1
f_1
f_0
f_1
f_0
f_1
f_0

Ved rekursiv beregning af f_n beregnes f_1 i alt f_n gange.

Sortering

Merge sort



1 2 3 5 6 7 8 9

procedure mergesort($L = a_1, \ldots, a_n$ liste af tal)
if $n > 1$ **then**
$m := \lfloor \frac{n}{2} \rfloor$
$L_1 := a_1, \ldots, a_m$
$L_2 := a_{m+1}, \ldots, a_n$
$L :=$ merge(mergesort(L_1), mergesort(L_2))

{ L er sorteret i ikke-aftagende rækkefølge}

Lemma 1 For at flette to sorterede lister med henholdsvis m og n elementer bruges højst $m + n - 1$ sammenligninger.

Flettet liste har $m + n$ tal.
Hvert tal fremkommer ved en sammenligning
undtagen hvis en af listerne er tom.

Sætning 1. Antal sammenligninger, der i alt bruges af Mergesort for at sortere en liste med $n = 2^k$ elementer med er højst

$\log n = k$

$$2^k(k-1)+1 = n(\log(n)-1)+1.$$

Mergesort har kompleksitet $O(n \log(n))$.

Bevis ved induktion efter k.

Basisskridt ($k=0$, $n=1$), 0 sammenligninger.

$$2^0(0-1)+1 = 1\cdot(-1)+1 = 0$$

$(k=1,\ n=2)$ 1 sammenligning

$$2^1(1-1)+1 = 0+1=1$$

Induktionsskridt Lad $l \geq 0$ og antag sætning er sand for $k=l$

Vist: $k=l+1$

Liste med $2^{l+1} = 2\cdot 2^l$ tal

Deles i to lister med 2^l tal hver

Sorter første liste: $\leq 2^l(l-1)+1$ (antagelse)

Sorter anden liste: $\leq 2^l(l-1)+1$

Flætning $\leq 2^l + 2^l - 1$

I alt:

$$2^l(l-1)+1 + 2^l(l-1)+1 + 2^l + 2^l - 1 =$$

$$2\cdot 2^l(l-1) + 2\cdot 2^l + 1 = 2^{l+1}(l-1) + 2^{l+1} + 1 =$$

$$2^{l+1}\cdot l + 1 = 2^{l+1}((l+1)-1) + 1$$