$$T(n) = \begin{cases} 1 & \text{når } n = 1 \\ T(n-1)+1 & \text{når } n > 1 \end{cases} \tag{2.1}$$

når vi lar tidsforbruket av ting som er $O(1)$, være 1. Da kan vi lage en tabell som viser tidsforbruket for ulike verdier av n, se tabell 2.1.

Tabell 2.1: Tidsforbruk, lineær rekursjon

n	T (n)
1	$T(1) = 1$
2	$T(2) = T(1)+1 = 1+1 = 2$
3	$T(3) = T(2)+1 = 2+1 = 3$
4	$T(4) = T(3)+1 = 3+1 = 4$
n	$T(n) = T(n-1)+1 = n$

Av tabellen ser vi at $T(n) = n$, slik at tidskompleksiteten blir $\Theta(n)$, det vi kaller lineær tidskompleksitet.

Et formelt bevis for dette er gitt under. Det er et induksjonsbevis, og starter med å bevise at basis er sann. Deretter antar vi at påstanden er sann for $n-1$, og beviser at da må den være sann for n. Og når disse to delene er bevist, kan vi slutte at påstanden må være sann for alle n fra og med basis og opp til uendelig.

Påstand: $T(n) = n$ for alle $n \geqslant 1$

Bevis Basis er OK siden $T(1) = 1$. For induksjonstrinnet har vi at vi skal anta at påstanden gjelder for $n-1$, dvs. at $T(n-1) = n-1$. Da kan vi bruke likning 2.1 til å finne at $T(n) = T(n-1)+1 = (n-1)+1 = n$, noe som stemmer med påstanden. Dermed er påstanden bevist. ■

Oppgaver

2.1–1 x^n kan defineres ved $x^n = \begin{cases} 1 & \text{når n=0} \\ x \cdot x^{n-1} & \text{når n>0} \end{cases}$. Programmer dette ved hjelp av rekursjon, og kontroller at svarene blir riktige.

2.1–2 Et palindrom er et ord eller setning som blir det/den samme uansett om vi leser det/den fra venstre mot høyre, eller fra høyre mot venstre. Eksempler på palindromer er «å», «tut», «madam» og «redder». Lag en rekursiv algoritme som finner ut om et ord er et palindrom.

2.1–3 Hva er feilen med følgende rekursive definisjon (oppslag i ei ordliste): Rekursjon: Se rekursjon.

2.1–4 Lag en iterativ versjon av oppgave 2.1–1.

Analyse

I algoritme 2.10 på forrige side har vi n rekursive kall og får følgende tidsforbruk når vi for enkelhets skyld glemmer konstantleddet 1:

$$T(n) = \begin{cases} 1 & \text{når } n = 1 \\ nT(n-1) & \text{når } n > 1 \end{cases} \tag{2.6}$$

Tabell 2.5: Tidsforbruk, n rekursjoner av størrelse $n-1$

n	**T (n)**
1	$T(1) = 1$
2	$T(2) = 2T(1) = 2$
3	$T(3) = 3T(2) = 6$
4	$T(4) = 4T(3) = 24$
5	$T(5) = 5T(4) = 120$
n	$T(n) = n!$

Tabell 2.5 viser at algoritmen har tidskompleksitet $\Theta(n!)$

Oppgaver

2.2–1 Bruk induksjon til å bevise likning 2.3 og 2.4.

2.2–2 Tegn de rekursive kallene som blir produsert ved utførelse av hanoi(4,'A','B','C');

2.2–3 x^n kan defineres ved $x^n = \begin{cases} 1 & \text{når } n = 0 \\ x \cdot \left(x^2\right)^{\frac{n-1}{2}} & \text{når } n \text{ er et oddetall} \\ \left(x^2\right)^{\frac{n}{2}} & \text{når } n \text{ er et partall} \end{cases}$

Programmer dette, og sammenlikn tidsforbruket ved stor n med det fra oppgave 2.1–1. Kan du forklare forskjellen?

2.2–4 Lag en iterativ versjon av oppgave 2.2–3.

2.2–5 Lag en rekursiv versjon av den iterative versjonen av Fibonacci-tallene. Dette skal bli en lineær rekursjon, og skal kalles på følgende måte for å finne fib(n): res = fiblin(1, 0, n) ; Hint: Lag følgende metodehode: int fiblin(int a, int b, int n), og la a og b være to påfølgende Fibonacci-tall i hvert rekursjonsnivå, slik at de fungerer omtrent som de interne variablene i den iterative versjonen.

2.2–6 Finn en algoritme som produserer alle delmengdene til en gitt mengde. (En delmengde av en mengde er en mengde som inneholder null eller flere av de samme elementene som mengden består av, f.eks. er a, c en delmengde av a, b, c, d.)