1 Tidskomplexitet

1.1 Introduktion

I datorvärlden är framtagandet av effektiva algoritmer en viktig sak. Det forskas mycket på att göra algoritmer så effektiva som möjligt. Ett mått på en ’bra’ algoritm är hur mycket tidsåtgången ökar då datamängden ökar. I sorteringssammanhang brukar det vara hur tiden för sortering beror på hur lång listan som ska sorteras är. Om tiden som det tar att sortera, T, beror kvadratiskt på listans längd, n, kan man tex skriva $T(n)=n^2$ (en potensfunktion). Urvalssortering har precis detta beroende. Om en lista med längden n_1 tar tiden T_1 för sortering, hur lång tid T_2 tar det för en dubbelt så lång lista att bli sorterad? Jo det tar

$$T_2=T(2n_1)=(2n_1)^2=4n_1^2=4T_1.$$

Dvs det tar 4 gånger så lång tid.

En snabb algoritm för sortering kan ha $T(n)=n\log_2(n)$. För en dubbelt så lång lista får man

$$T_2=T(2n_1)=2n_1\log_2(2n_1)=2n_1\left(\log_2(2)+\log_2(n_1)\right)=$$
$$2n_1\log_2(2)+2n_1\log_2(n_1)\approx$$
$$2n_1+2T_1$$

Här ökar tiden med en faktor 2, n_1 är en ’konstant’ som beror på var vi började jämförelsen. Denna algoritm ökar i allmänhet långsammare än den för Urvalssortering.

Beräkningsexempel. Om det tar 10 s att sortera 10 000 tal, hur lång tid tar det att sortera 20 000 tal? Enligt $T(n)=n^2$ så gäller

$$\frac{20000^2}{10000^2}=\frac{2^2}{1^2}=4$$

således 40 s. I det andra fallet

$$\frac{20000\cdot\log_2(20000)}{10000\cdot\log_2(10000)}=2\frac{\log_2 2+\log_2 10^4}{\log_2 1+\log_2 10^4}=$$
$$2\frac{1+13,3}{0+13,3}\approx 2,2$$

således 22 s. Och skillnaderna mellan de 2 metoderna blir bara större och större.

1.2 Orsaker

Varför blir det så här? Tiden för sortering beror i huvudsak på hur många matematiska operationer som behöver utföras. Alla operationer tar tid. Man behöver titta på hur många additioner, subtraktioner, multiplikationer, divisioner som utförs, och eventuellt även mer komplicerade operationer som exponentiering. Vi genomför inte en sådan analys här.

Detta leder också till en viss statistisk variation. Om man har en algoritm som är bra så kanske man inte behöver göra så många genomgångar eller operationer om listan redan är nästan sorterad; då blir det viktigt att man slumpar flera listor och mäter tidsåtgången för sortering och beräknar medelvärde.
För andra algoritmer, som Urvalssortering, så går man alltid igenom hela listan oavsett om den redan är sorterad eller inte; men man behöver göra färre antal uppdateringar av minst_i och minst. Även detta ger naturligtvis en statistisk variation i tidsåtgången men den dominerar inte.
Sedan är datorn kanske upptagen med andra processer och de kanske stör tidsmätningen. Som användare kan du oftast inte styra processorn helt.