Sammenlikningsbasert sortering

Petter Kristiansen

2008-10-07

Sammendrag

Et lite notat om antall sammenlikninger som trengs for å sortere n tall, om man bruker såkalte *sammenlikningsbaserte sorterings-algoritmer*, altså ikke slikt som bøtte- og radix-sortering eller liknende. Dette notatet er kun å regne som orienteringsstoff i INF4130 høsten 2008.

1 Innledning

Vi skal vise hvor mange sammenlikninger som minst må gjøres for å sortere n tall. Fra før vet vi at f.eks BubbleSort har kjøretid $O(n^2)$, og altså at det *holder* med n^2 sammenlikninger; men hvor mange er vi *nødt til* å gjøre for å kunne sortere enhver mulig sekvens av n tall? Som kjent er det $n! = n * (n\text{-}1) * (n\text{-}2) * \ldots * 2 * 1$ mulige sekvenser av heltallene 1, 2, …, n. Vi viser at $\Omega(n \log n)$ sammenlikninger trengs i verste fall (worst case). (Notasjonen $\Omega(f(n))$ brukes for å angi en nedre beskrankning, på samme måte som *O-notasjon* brukes til å angi en øvre beskrankning for en funksjon.) Det kan også vises at $\Omega(n \log n)$ sammenlikninger trengs i gjennomsnitt, men det skal vi ikke se på her.

2 En nedre grense

Vi ser nå på sammenlikningsbaserte sorteringsalgoritmer, algoritmer som bare opererer på inputsekvensen (tallene som skal sorteres) ved å sammenlikne par av tall, og eventuelt bytte om på disse. Vi skal bruke følgende definisjon:

Definisjon 2.1 En sammenlikningsbasert sorteringsalgoritme får som input en sekvens $I = [i_1, i_2, \ldots, i_n]$ av elementene $\{1, 2, \ldots, n\}$ som skal sorteres, og kan kun skaffe seg informasjon om disse elementene ved å sammenlikne par for par av dem. Hver sammenlikning (av typen «Er $i_k < i_l$?») besvares med TRUE eller FALSE og teller som ett tidssteg. Algoritmen bytter ikke om på elementene underveis, men gir oss en permutasjon av input-sekvensen hvor elementene er sortert – en rekkefølge vi skal lese input-sekvensen i for å få tallene sortert.

BubbleSort og QuickSort er eksempler på sammenlikningsbaserte sorterings-algoritmer, mens BucketSort ikke er det.

Vi viser følgende:

Teorem 2.1 Enhver sammenlikningsbasert sorteringsalgoritme må gjennomføre $\Omega(n \log n)$ sammenlikninger for å sortere n elementer i verste fall. Spesifikt vil det for enhver sammenlikningsbasert sorteringsalgoritme $\boldsymbol{A}$, for alle $n > 2$ finnes en input I av størrelse n hvor $\boldsymbol{A}$ gjør minst $\log n! = \Omega(n \log n)$ sammenlikninger for å sortere I.

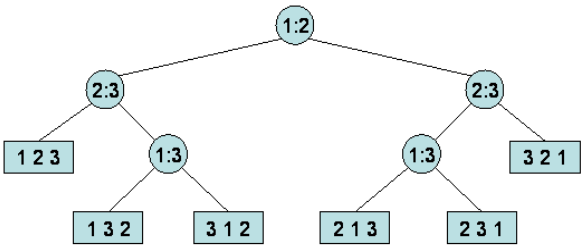
Bevis. Vi kan ikke anta noe om algoritmens virkemåte, utover at den baserer seg på sammenlikninger av elementene i input, la input være $I = i_1, \ldots, i_n$. Sammenlikningene som gjøres kan vi representere som et tre – et *desisjonstre*:

- Hver sammenlikning algoritmen gjør mellom elementene i_k og i_l gir oss en intern-node $(i_k{:}i_l)$.
- Venstre subtre av denne intern-noden representerer påfølgende sammenlikninger som gjøres hvis $i_k < i_l$. Høyre subtre av denne intern-noden representerer påfølgende sammenlikninger som gjøres hvis $i_k > i_l$.
- Løvnodene vil bestå en permutasjon $r_1, r_2, \ldots, r_n$ av $\{1, 2, \ldots, n\}$, som indikerer at

  $$i_{r_1} < i_{r_2} < \cdots < i_{r_n} .$$

  Dvs. at den sorterte rekkefølgen av elementene i I blir definert av r_i-ene. Det er $n! = n * (n\text{-}1) * (n\text{-}2) * \ldots * 2 * 1$ mulige permutasjoner av heltallene $\{1, 2, \ldots, n\}$, og altså $n!$ mulige slike løvnoder.

Figuren under viser et eksempel på et slikt desisjonstre.



Om sammenlikningene algoritmen gjør (fra roten og nedover) er slik at $i_1<i_2$, $i_2>i_3$ og $i_1>i_3$, vil sortert rekkefølge av elementene i I være i_3, i_1, i_2. (Tredje løvnode fra venstre.)

Et slikt binært desisjonstre med $n!$ løvnoder, må ha høyde minst

$$\begin{aligned}\log(n!) &= \log(n) + \log(n-1) + \cdots + \log 2 + \log 1 \\ &= \Omega(n \log n).\end{aligned}$$

Stirlings approksimasjon gir oss et litt nøyaktigere mål

$$\log(n!) = n \log n - \frac{n}{\log_e 2} + \frac{\log n}{2} + O(1).$$

■