Nedre grænse for sammenligningsbaseret sortering

Søren Dahlgaard

Datalogisk Institut, Københavns Universitet

Diskret Matematik og Algoritmer, 2015

Motivation

Alle de sorteringsalgoritmer, vi har set, fungerer ved at bruge sammenligninger.
Ingen af dem er hurtigere end $\Theta(n \log n)$ i værste tilfælde.

- Insertion sort $O(n^2)$
- Merge sort $\Theta(n \log n)$
- Heap sort $\Theta(n \log n)$

Kan man gøre det bedre? Er der en grund til at vi hele tiden ser $\Theta(n \log n)$?

Sagt anderledes: Er det mulig at lave en algoritme, der er bedre end $\Omega(n \log n)$ i det værste tilfælde, hvis den kun bruger sammenligninger og ombytninger?

Problemformulering

Hvad er det vi vil argumentere?

Vi ønsker at vise følgende sætning:

Theorem 1

Enhver korrekt sorteringsalgoritme, der afgør den korrekte rækkefølge af elementerne udelukkende på baggrund af sammenligninger, skal bruge $\Omega(n \log n)$ *sammenligninger i værste tilfælde.*

Baggrundsstof

Observationer

Ethvert input $(x_1, \ldots, x_n)$ har $n! = n \cdot (n-1) \cdots 1$ permutationer

Vi ved ikke på forhånd hvilke(n) permutation(er) der er sorterede.

Algoritmen skal kunne outputte enhver mulig permutation for at være rigtig.

Eksempel: Hvis algoritmen ikke kan outputte permutationen $(x_n, x_{n-1}, \ldots, x_1)$ kan den ikke sortere listen $(5, 4, 3, 2, 1)$ korrekt.

Baggrundsstof

Observationer fortsat

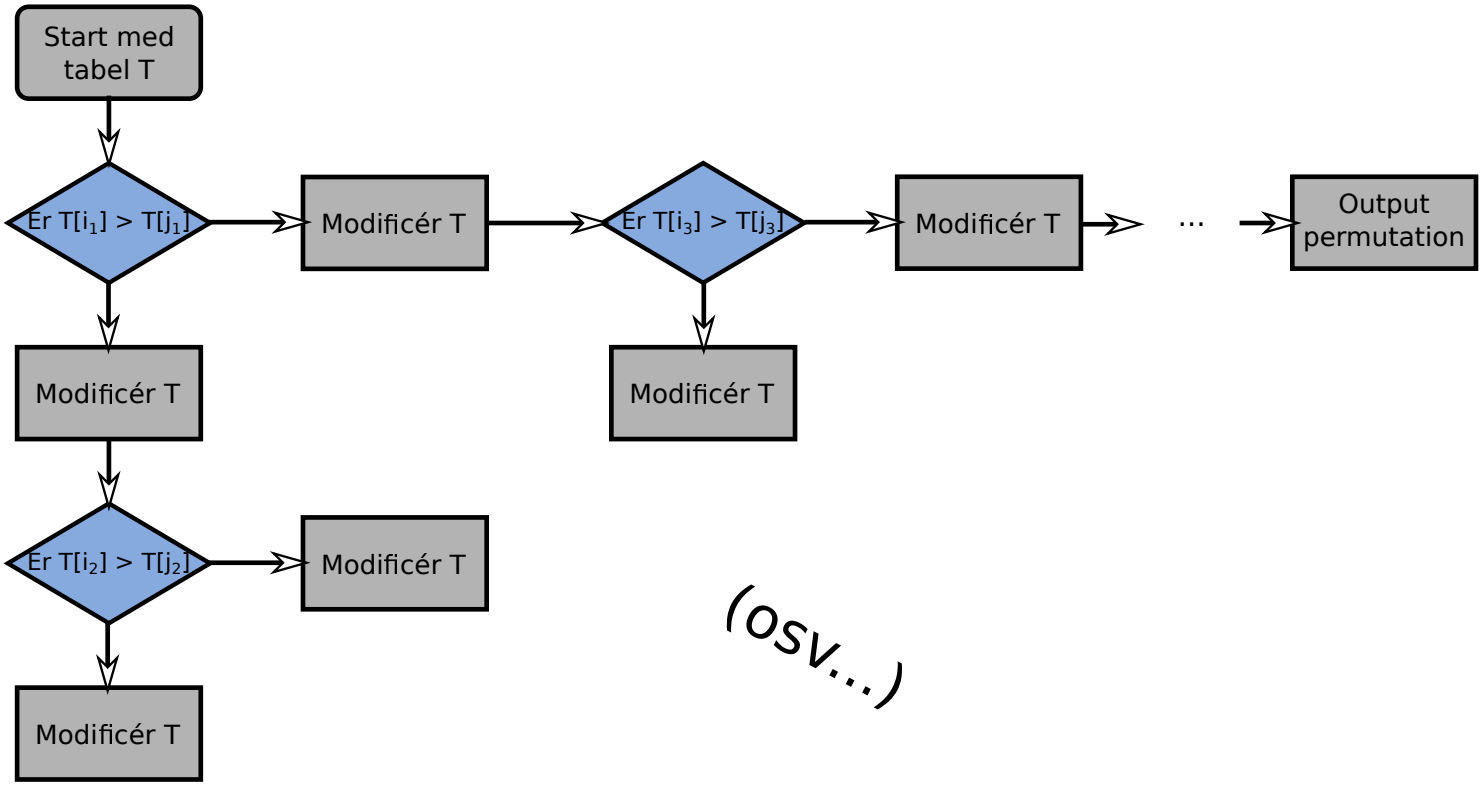
Hvis en algoritme til at sortere 3 tal ikke kan give alle $3! = 6$ permutationer vil der være (mindst) en af følgende lister den ikke kan sortere!

Input	**Sorteret permutation**
$(1, 3, 5)$	(x_1, x_2, x_3)
$(1, 5, 3)$	(x_1, x_3, x_2)
$(3, 1, 5)$	(x_2, x_1, x_3)
$(3, 5, 1)$	(x_3, x_1, x_2)
$(5, 1, 3)$	(x_2, x_3, x_1)
$(5, 3, 1)$	(x_3, x_2, x_1)

Baggrundsstof

Hvordan ser en algoritme ud?

Enhver sammenligningsbaseret sorteringsalgoritme kan beskrives således:



Baggrundsstof

Hvordan ser en algoritme ud? (fortsat)

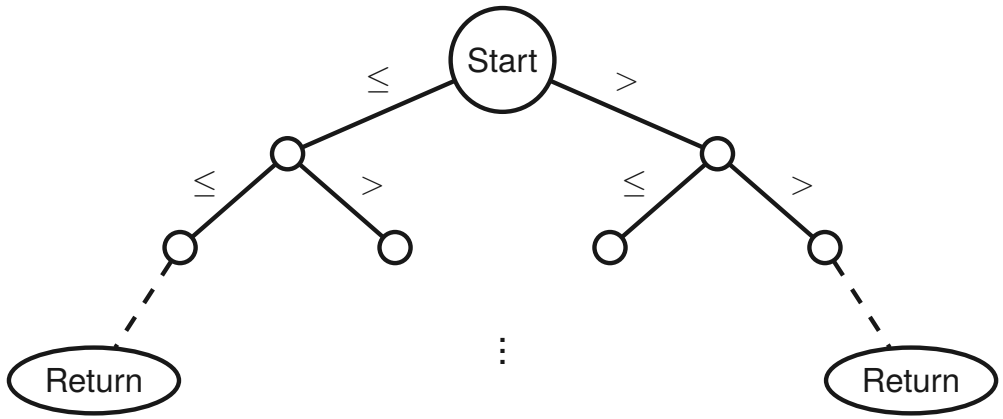
Sagt med andre ord gør algoritmen følgende:

1. Sammenligner to elementer $T[i_1]$ og $T[j_1]$
2. Ændrer tabellen T afhængig af resultatet af sammenligningen.
3. Sammenligner to (potentielt) andre elementer $T[i_2]$ og $T[j_2]$.
4. Ændrer tabellen T afhængig af resultatet af sammenligningen.
5. ...
6. Returnerer den resulterende tabel T.

Træ repræsentation

Hvordan beviser vi Theorem 1?

Ved hver sammenligning forgrener algoritmen sig i to:
Svarer til et binært træ! (blad = algoritmen returnerer)



En vej fra rod til blad i træet svarer til en eksekvering af algoritmen på et specifikt input! Derfor er længden ≈ køretid.

Nedre grænse

At sætte det hele sammen

Vi ved følgende:

- En algoritme i sammenligningsmodellen svarer til et binært træ.
- Hvert blad er en mulig løsning.
- Hver mulig løsning er en permutation.
- Worst-case køretiden er længden af den længste vej fra roden til et blad (dvs. højden af træet).
- Der er mindst $n!$ løsninger (blade).
- Et binært træ med højde h har højst 2^h blade.

$$2^h \geq \#\text{antal blade} \geq n! \Rightarrow h \geq \log_2(n!) = \Omega(n \log n)$$

Nedre grænse

Yderligere forklaring og konklusion

Da en algoritme svarer til et sådant beslutningstræ kan vi altså gøre følgende:

1. Kig på beslutningstræet der svarer til en given sorteringsalgoritme.
2. Hvert blad svarer til en inputpermutation. Bemærk at nogle permutationer kan have flere blade!
3. Find den permutation hvis øverste blad har størst dybde (må være mindst $\log_2(n!)$).
4. Giv algoritmen denne permutation som input. Nu skal algoritmen bruge $\Omega(n \log n)$ sammenligninger for at sortere permutationen!

Lige meget hvilken algoritme (beslutningstræ) vi får kan vi således finde et input, der kræver $\Omega(n \log n)$ sammenligninger!

Opsamling

Vi har nu bevist at en enhver sammenligningssortering skal bruge $\Omega(n \log n)$ sammenligninger i værste tilfælde. Hovedidéerne var følgende:

1. Modellér algoritmen som et beslutningstræ - Kan vi gøre pga. sammenligningsmodellen.
2. Sig noget om størrelsen på træet - Kan vi gøre fordi hvert blad svarer til en permutation.
3. Konkludér at højden (køretiden) på træet er stor - Kan vi gøre fordi træet er binært.

Øvelse: Brug samme metode til at vise at sammenligningsbaseret søgning kræver $\Omega(\log n)$ sammenligninger i værste tilfælde.