Eksamensdisposition - Del og hersk

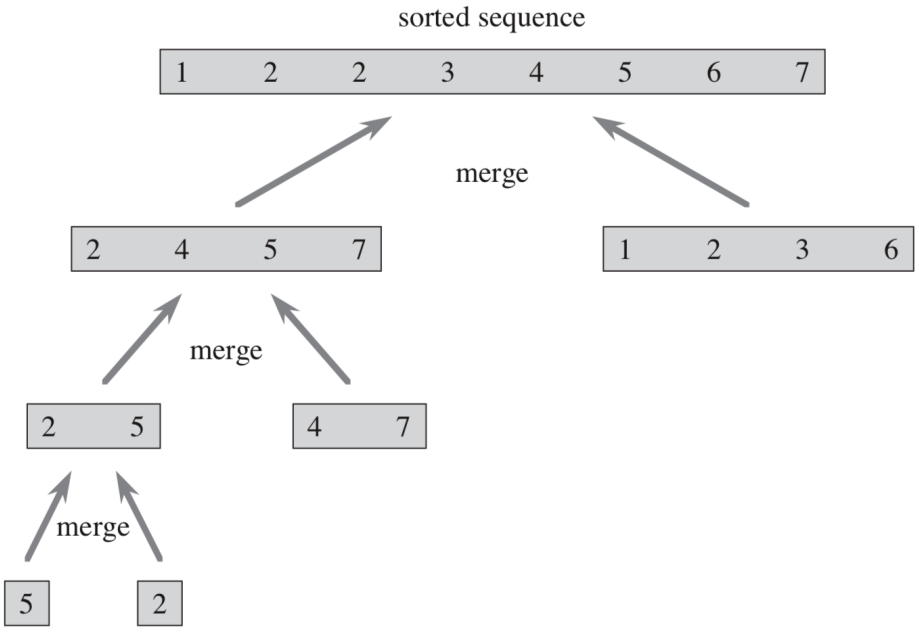
Søren Mulvad, rbn601

3. april 2019

- **Paradigmet**
- **Håndkørsel af `MergeSort`**
- **Rekursionsrelationer**
  - Rekursionstræer
  - Substitutionsmetoden
  - Kort om Mastermetoden
- **Bevis for lower bound for sortering**

1 Divide and Conquer (Del-og-hersk)

- **Paradigmet**
  - *Del* problemet ind i en mængde af delproblemer som er mindre instanser af det samme problem.
  - *Hersk* (løs) delproblemerne ved at løse dem rekursivt. Hvis delproblemerne er tilpas små, så løs dem bare "direkte".
  - *Kombinér* løsningerne til delproblemerne sammen til én løsning til det oprindelige problem.
  - Minder en smule om Dynamisk Programmering, men forskellen er, at problemerne ikke har optimal delstruktur, altså der er ikke overlappende delproblemer.
  - Eksempler på algoritmer der benytter det er `QuickSort`, `Find-Maximum-Subarray` og `MergeSort`, som jeg vil gå mere i dybden med nu.
- **Håndkørsel af `MergeSort`**



Figur 1: Tegn et træ ala dette, hvor du så kun viser ned i venstre side. Sørg for at holde det 8 elementer langt.

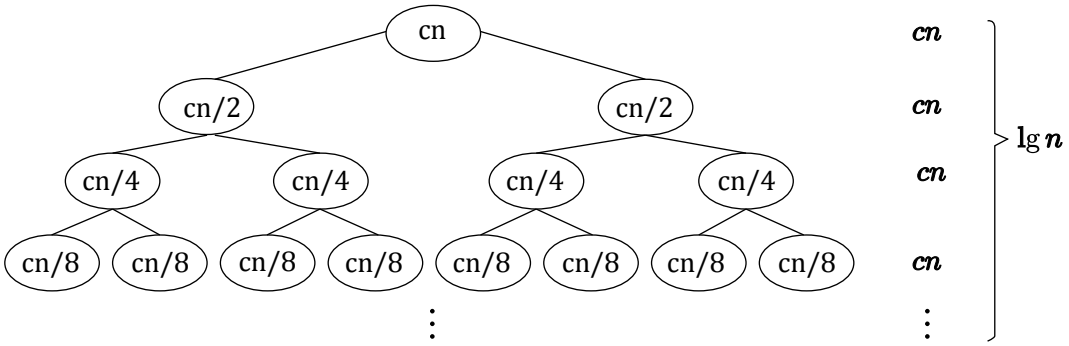
Jeg vil vise det på en 2-talspotens, for det gør det lidt nemmere at illustrere, men princippet ville være det samme for andre tal. Del arrayet op i to (ca.) lige store dele, kald `Merge-Sort` på disse og `Merge` til sidst de nu sorterede delarrays. Algoritmen kører i $O(n \lg n)$ tid, og det vil jeg nu bevise.

- **Rekursionsrelationer**
  Rekursionsrelationer er de typer relationer vi typisk vil få ved del og hersk algoritmer. Som et eksempel ses her rekursionsrelationen for `MergeSort`, hvor vi undlader at skrive casen når $n = 1$ eksplicit og ignorer, at vi egentlig skal tage floor og ceiling, altså tager udgangspunkt i den case hvor vi har en 2-talspotens. Vi får denne rekursionsrelation, da vi åbner to nye delproblemer der er af halv størrelse af det originale, og derudover til sidst skal samle det originale:

$$T(n) = 2T(n/2) + cn$$

- **Rekursionstræer**
  Kan, hvis man er meget præcis, bruges til at bevise en rekursionsrelation. Bruges dog oftest til

at komme på et godt gæt man så kan bevise med substitutionsmetoden. Her ses rekursionstræet vi får for `MergeSort` idet vi bruger ovenstående rekursionsrelation:



Figur 2: Rekursionstræ for `MergeSort`

Vi ser, at hvert niveau i træet bidrager med cn og der er $\lg n$ af disse led. Derfor gætter vi på, at køretiden af `MergeSort` er $O(n \lg n)$. Vi gætter på dette:

$$T(n) = 2T(n/2) + n \qquad \text{har køretid} \qquad O(n \lg n)$$

- **Substitutionsmetoden**

  1. Gæt på en løsning (enten ud fra erfaring eller ved at tegne et rekursionstræ)
  2. Bevis din løsning er sand ved hjælp af induktion (OBS: Vær opmærksom på vi ikke kan være så ligeglad med konstanter som vi ellers tit er med O-notation). Hvis vi løber ind i et problem kan det ofte være en hjælp at trække et led af en lavere orden fra.
  3. Hvis du stadig ikke kan bevise det, så lav et nyt gæt og prøv igen.

Til dette bevis skulle vi normalt bruge stærk induktion, men da vi antager det er en 2-talspotens er det ikke nødvendigt.

Base case:
Vi skal nu vise $T(n) \leq cn \lg n$, hvor vi antager $T(1) = 1$. Vi ser, at vi ikke kan vise det for $n = 1$ da vi så får $cn \lg n = 0$ da $\lg 1 = 0$.

For $n = 2$:

$$T(2) = 2T(\lfloor 2/2 \rfloor) + 2 = 2T(1) + 2 = 4 \leq c2 = c2 \lg 2$$

Hvilket er sandt for $c \geq 2$. Nu går vi til induktionssteppet:

Induktionsstep:

$$\begin{aligned} T(n) &\leq 2c \frac{n}{2} \lg(n/2) + n && (1)\\ &= cn \lg(n/2) + n && (2)\\ &= cn \lg n - cn \lg 2 + n && (3)\\ &= cn \lg n - cn + n && (4)\\ &\leq cn \lg n && (5) \end{aligned}$$

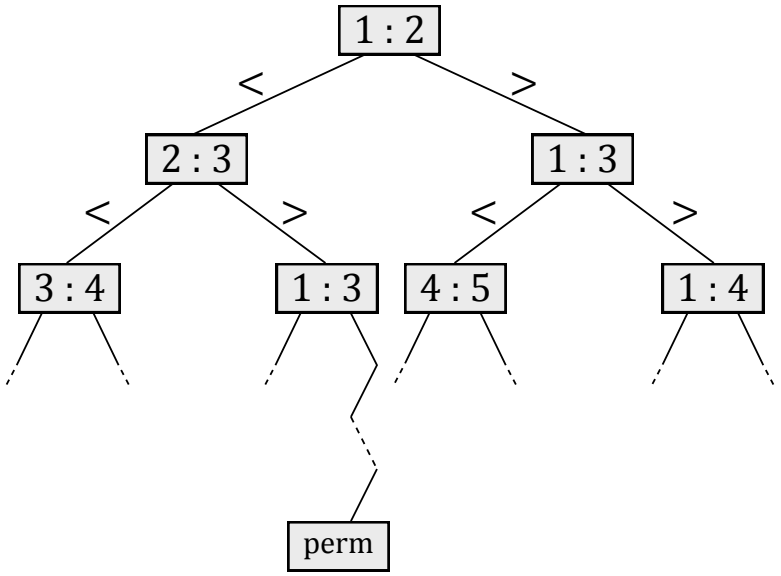
I Eq. (1) benytter vi blot vores induktionsantagelse.
I Eq. (2) ganger vi 2 ind i brøken.
I Eq. (3) deler vi logaritmen op jf. logaritmeregneregler.
I Eq. (4) fjerner vi multiplikationsleddet $\lg 2 = 1$.
I Eq. (5) ved vi der gælder, at $-cn + n \leq 0$ så længe $c \geq 1$.

- **Mastermetoden**
  Lad $a \geq 1$ og $b > 1$ være konstanter samt $f(n)$ en funktion, og lad $T(n)$ være defineret ved rekursionsrelationen:
  $$T(n) = aT(n/b) + f(n)$$
  hvor n/b kan være være både $\lfloor n/b \rfloor$ eller $\lceil n/b \rceil$. Så er der tre forskellige cases, og vi kan næsten altid beregne $T(n)$ asymptotiske grænse.

- **Lower bound for comparison sort**
  Vi antager for at gøre det nemmere, at der ikke er elementer som er ens. Derudover vil vi kun bruge operationen $a < b$ til at få informationer om to elementers relation, da alle andre giver ækvivalent information.

  Antag vi får et input array A med n elementer. Nu anvender vi en comparison sort S på denne. Det S gør, er at permutere elementer i A til en sorteret liste.

  Denne permutation er unikt defineret ud fra de sammenligninger algoritmen har foretaget. Alle disse udfald kan vi repræsentere i et beslutningstræ. Alle simple veje fra rodknuden til et blad er alle de forskellige permutationer der findes.



  Lad nu antallet af blade være b og højden h. Såfremt træet er perfekt balanceret kan vi presse 2^h blade ind i træet. Og vi ved også, at alle permutationer forekommer i bladene, derfor må $b \geq n!$ for ellers ville der ikke være plads til alle permutationer i bladene. Derfor får vi:

  $$2^h \geq b \geq n!$$

  Vi kan tage 2tals-logaritmen hvorved vi får:

  $$h \geq \lg(n!) = \Omega(n \lg n)$$

  Herved har vi altså bevist, at der er et input af længde n som kræver at der bliver foretaget $h = \Omega(n \lg n)$ sammenligner.