Institut for Matematik og Datalogi
Syddansk Universitet, Odense

6. oktober 2003
finie02@imada.sdu.dk

DM02 – Opgaver fra ugeseddel 5

Øvelsesopgaver 30/9, 1/10 og 2/10

1. Cormen et al. 2.3-1
   (udeladt med vilje)

   2.3-2
   Omskriv MERGE prodceduren (s. 29) så den ikke bruger 'skildvagter' (eng. 'sentinels'), men istedet stopper når enten R eller L har alle sine elementer kopieret tilbage til A, og derfra kopierer resten af det andet array tilbage til A.
   Skildvagterne i denne opgave er elementerne på hhv. plads n_1+1 og plads n_2+1 som er sat til ∞, som sørger for at kopieringen aldrig når videre end til det sidste element i hhv. array L og R.

   Et forslag til omskrivning fra linie 12-13 i den oprindelige algoritme:

```
for k <- p to r //linie 12
  if i == n1 + 1
    //kopiér resten af R til A
    while j < n2 + 1
      A[k] <- R[j]
      j ++
      k ++
  else if j == n2 + 1
    //kopiér resten af L til A
    while i < n1 + 1
      A[k] <- L[j]
      i ++
      k ++
  else
    do if L[i] <= R[j]
    //fortsæt fra linie 14 i algoritmen s. 29
```

   Husk at kodeomskrivninger kan gøres på mange måder. Brug den omskrivning som du selv synes giver det lettest læselige billede af hvad algoritmen foretager sig.

   2.3-4.
   Insertion sort kan udtrykkes som en rekursiv procedure
   (oversat fra Cormen s. 37):

   > For at sortere A[1...n], skal vi først rekursivt sortere A[1...n-1] og derefter indsætte A[n] i det sorterede array A[1...n-1]

   Skriv rekursionsligningen for køretiden af denne rekursive version af insertion sort.

$$T(n) = \begin{cases} \Theta(1) & ,n = 1 \\ T(n-1) + O(n) & ,ellers \end{cases}$$

I den første linie angiver vi, at hvis vi kun har ét element, dvs. $n = 1$, er vores array allerede sorteret, så vi udfører kun et konstant antal operationer, altså $\Theta(1)$. I anden linie er $T(n-1)$ den rekursive sortering af resten af array'et ($A[1...n-1]$), mens $O(n)$ er tiden det tager at sætte $A[n]$ ind på sin korrekte plads i $A[1...n-1]$.

2. Cormen et al. 2-1.
I den asymptotiske notation for Merge-sort gemmer sig nogle konstanter, som ved små input gør Merge-sort langsommere end tilsvarende input sorteret med Insertion-sort.
(Eksempelvis vil $f(n) = 2nlg(n) + 5$ være større end $g(n) = n^2 + 1$ for små n)

Vi overvejer nu en modificeret Merge-sort, hvor n/k dellister af længde k bliver sorteret vha. Insertion-sort og derefter flettet vha. den almindelige Merge fra Merge-sort.

(a) Vis at de n/k dellister af længde k kan sorteres med Insertion-sort i tid $\Theta(nk)$ i worst-case.
Insertion-sort bruger i worst-case $T_{Isort}(n) \in \Theta(n^2)$, så ved n/k lister af længde k skal vi i alt bruge:

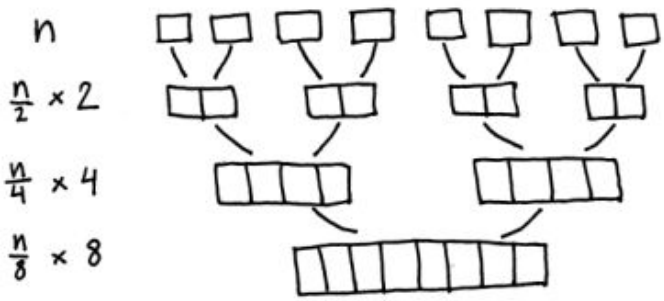
$$antallister \cdot T_{Isort}(listelængde) = n/k \cdot T_{Isort}(k) = n/k \cdot \Theta(k^2) = \Theta(nk)$$

Hvorfor kan man bare 'gange ind i' et udtryk med theta?
Det er fordi at $f(n) \in \Theta(g(n))$ bare betyder at funktionen $f(n)$ kan skrives som en konstant gange $g(n)$ + evt. nogle termer der er af mindre grad end $g(n)$, dvs. $h(n) \in o(g(n))$. Altså:

$$f(n) \in \Theta(g(n)) \Leftrightarrow f(n) = c \cdot g(n) + h(n)$$

Så når vi ganger med en funktion eller en værdi og igen får et udtryk på ovenstående form, med andre c og $h(n)$, så har vi stadig at $f(n) \in g(n)$.

(b) Vis at dellisterne kan merges i $\Theta(nlg(n/k))$ worst-case.
Vi har n/k lister og vores Merge-procedure kører i $\Theta(n)$ (s.28). Hvis vi skal flette n elementer, skal vi først køre Merge på elementerne to og to, derefter på de resulterende lister to og to, indtil alle elementer er flettet til et enkelt array. Vi kan samle dellister to og to ialt $lg(n)$ gange.



I første niveau vil vi kalde Merge $n/2$ gange for at få lister af størrelse 2, derefter kaldes Merge på $n/4$ lister af størrelsen 4, etc.

Altså vil vi altid have $\Theta(n)$ elementer for hver niveau af kald til Merge. Derved får vi den samlede kompleksitet som antal niveau af kald til Merge ganget med kompleksiteten af Merge

$$lg(n/k) \cdot \Theta(n) = lg(n/k) \cdot cn = cnlg(n/k) \in \Theta(nlg(n/k))$$

(c) Givet at den modificerede algoritme kører i $\Theta(nlg(n/k) + nk)$ worst case, hvad er den største asymptotiske (Θ notation) værdi af k som funktion af n, så den modificerede algoritme stadig har samme asymptotiske køretid som standard Merge sort?

Standard Merge sort $\in \Theta(nlg(n))$ (s. 36)

1. observation $k < n$
hvis $k = n$ bruger vi Insertion sort på hele A $\Rightarrow \Theta(n^2)$ i worst-case.

2. observation $k \geq 1$
da vi maksimalt kan lave n dellister af 1 element hver.

3. observation $nlg(n/k) < nlg(n)$, når $k < n$
Så vi behøver kun at finde ud af hvornår $nk \leq nlg(n)$, og det er klart at det er opfyldt for $k \leq lg(n) \Rightarrow \Theta(nlg(n) + nlg(\frac{n}{lg(n)})) \in \Theta(nlg(n))$

(d) Hvordan skal k-værdien vælges i praksis?
I praksis må vi ved måling konstatere ved hvilke værdier n for array-længden at Insertionsort kører hurtigere end Merge-sort, på netop dén maskine vi vælger at køre vores modificerede algoritme på. Altså skal vi vælge $k = n$ således at $Insertionsort(n) < Mergesort(n)$.

Denne opgave er lidt hypotetisk, da vi sjældent vil bruge tid på at optimere Mergesort, når vi generelt bare kan vælge en algoritme som har en god køretid og små skjulte konstanter.

3. Eksamen juni 96 opg. 1.
(udeladt med vilje)

4. Lad $W : \mathbb{N} \to \mathbb{N}$ være funktionen defineret ved

$$W(n) = \begin{cases} 0, & n = 0 \\ 1 + W\left(\lfloor \frac{n}{2} \rfloor\right), & n \geq 1 \end{cases}$$

Vis ved induktion over n, at $W(n)$ er monotont ikke-aftagende.
Dvs. vis, at $W(n) \leq W(n+1)$ for alle heltal $n \geq 0$.

Basis $n = 0$
$W(0) = 0$ ifølge definitionen
$W(0+1) = W(1) = 1 + W(\lfloor \frac{1}{2} \rfloor) = 1 + W(0) = 1 + 0 = 1$
Altså kan vi konkludere at $W(0) \leq W(0+1)$ er opfyldt

Induktion $n \geq 0$
Antag $W(k) \leq W(k+1)$ for $k \geq 0$. Vis at $W(k+1) \leq W(k+2)$.

Vi opdeler i to tilfælde, k lige og k ulige:

- k lige:

$$\begin{aligned} W(k+1) &= 1 + W(\lfloor \frac{k+1}{2} \rfloor) \\ \text{(da k lige)} &= 1 + W(\frac{k}{2}) \end{aligned}$$

$$\begin{aligned} W(k+2) &= 1 + W(\lfloor \frac{k+2}{2} \rfloor) \\ \text{(da k lige)} &= 1 + W(\frac{k}{2} + 1) \end{aligned}$$

$$\Leftrightarrow W(k+1) < W(k+2)$$

- k ulige:

$$\begin{aligned} W(k+1) &= 1 + W(\lfloor \frac{k+1}{2} \rfloor) \\ \text{(da k ulige)} &= 1 + W(\frac{k+1}{2}) \end{aligned}$$

$$\begin{aligned} W(k+2) &= 1 + W(\lfloor \frac{k+2}{2} \rfloor) \\ \text{(da k ulige)} &= 1 + W(\frac{k+1}{2} + 1) \end{aligned}$$

$$\Leftrightarrow W(k+1) = W(k+2)$$

Altså kan vi konkludere $W(k+1) \leq W(k+2)$ for alle k.

5. I sidste uge skrev I pseudokode for binær søgning. Det kunne f.eks. se sådan ud:

```
BINARYSEARCH(x, A, p, r)
    if p > r
        return −1
    q ← ⌊(p+r)/2⌋
    if x = A[q]
        return q
    else if x < A[q]
        return BINARYSEARCH(x, A, p, q − 1)
    else
        return BINARYSEARCH(x, A, q + 1, r)
```

I sidste uge argumenterede I blot for, at worst-case køretiden er $\Theta(\log n)$. I denne uge skal I give et formelt bevis: Opstil en rekursionsligning for worst-case tidskompleksiteten af algoritmen, og løs den. Se først på tilfældet, hvor længden af array'et kan skrives som $2^k - 1$, hvor k er et ikke-negativt heltal. Brug derefter opgave 4. ovenfor til udvide beviset til at gælde for en vilkårlig array-længde.

(udeladt med vilje)