Algoritmer ugeopgave 1
Genaflevering af Task 2 og 4

Nicklas Warming Jacobsen
Christian Enevoldsen
Simon Warg
Robert Rasmussen

D. 20. Maj 2014

Task 1

1: For $p(n) = 8p(n/2) + n^2$ gælder det at $p(n) = \Theta(n^3)$. Dette kan man se ved at bruge Theorem 4.1 case 1, hvor det skal gælde at for $\epsilon > 0$:

$$f(n) = O(n^{log_b a - \epsilon})$$

ϵ bliver valgt til $\epsilon = 1$

$$n^2 = O(n^{log_2 8 - 1}) \Leftrightarrow n^2 = O(n^2) \Downarrow$$
$$p(n) = \Theta(n^{log_2 8}) = \Theta(n^3)$$

2: For $p(n) = 8p(n/4) + n^3$ gælder det at $p(n) = \Theta(n^3)$. Dette kan man se ved at bruge Theorem 4.1 case 3, hvor det skal gælde at for $\epsilon > 0 \wedge c < 1$:

$$f(n) = \Omega(n^{log_b a + \epsilon}) \wedge a \cdot f(n/b) \leq c \cdot f(n)$$

ϵ bliver valgt til $\epsilon = 1,5$

$$n^3 = \Omega(n^{log_4 8 - 15})$$
$$\Updownarrow$$
$$n^3 = \Omega(n^3)$$

Vi vælger $c = 1/8$ og får:

$$8\left(\frac{n}{4}\right)^3 \leq \frac{1}{8} \cdot n^3$$
$$\Updownarrow$$
$$8\frac{n^3}{64} \leq \frac{1}{8} \cdot n^3$$
$$\Updownarrow$$
$$\frac{n^3}{8} \leq \frac{n^3}{8}$$

3: For $p(n) = 10p(n/9) + n \cdot log_2 n$ gælder det at $p(n) = \Theta(n^{log_9 10} log_2 n) \approx \Theta(n^{1,048} log_2 n)$. Dette kan man se ved at bruge Theorem 4.1 case 3, hvor det skal gælde at:

$$f(n) = \Theta(n^{log_b a})$$

$$n \cdot log_2 n = \Theta(n^{log_9 10})$$
$$n \cdot log_2 n = \Theta(n^{1,048})$$

Task 2

Opgave 1

Vi har $p(n) = p(\frac{n}{n}) + p(\frac{n}{3}) + n$, vi gætter at $P(n) = \Theta(cn) = \Theta(n)$.
Vi viser upper-bound via substitutions metoden:

$$\begin{aligned} p(n) &= p\left(\frac{n}{n}\right) + p\left(\frac{n}{3}\right) + n \\ &\leq \frac{cn}{2} + \frac{cn}{3} + n \\ &= \frac{3cn}{6} + \frac{2cn}{6} + n \\ &= \frac{5cn}{6} + n \\ &\leq cn \end{aligned}$$

Vi finder konstanten c:

$$\begin{aligned} \frac{5cn}{6} + n &\leq cn \\ 5cn + 6n &\leq 6cn \\ 5c + 6 &\leq 6c \\ 6 &\leq c \end{aligned}$$

På samme måde findes lower-bound:

$$\begin{aligned} p(n) &= p\left(\frac{n}{n}\right) + p\left(\frac{n}{3}\right) + n \\ &\geq \frac{cn}{2} + \frac{cn}{3} + n \\ &= \frac{3cn}{6} + \frac{2cn}{6} + n \\ &= \frac{5cn}{6} + n \\ &\geq cn \end{aligned}$$

Vi finder konstanten c:

$$\begin{aligned}
\frac{5cn}{6} + n &\geq cn \\
5cn + 6n &\geq 6cn \\
5c + 6 &\geq 6c \\
6 &\geq c
\end{aligned}$$

Opgave 2

Vi har at: $P(n) = \sqrt{n} \cdot p\left(\sqrt{n}\right) + \sqrt{n}$, vi gætter at $P(n) = \Theta(cn - 1) = \Theta(n)$
Vi viser upper-bound via substitutions metoden:

$$\begin{aligned}
P(n) &= \sqrt{n} \cdot p\left(\sqrt{n}\right) + \sqrt{n} \\
&\leq \sqrt{n} \cdot \left(c\sqrt{n} - 1\right) + \sqrt{n} \\
&= cn - \sqrt{n} + \sqrt{n} \\
&= cn
\end{aligned}$$

Vi har altså fundet at $P(n) = O(n)$ for $c > 0$.
På samme måde finder vi lower-bound:

$$\begin{aligned}
P(n) &= \sqrt{n} \cdot p\left(\sqrt{n}\right) + \sqrt{n} \\
&\geq \sqrt{n} \cdot \left(c\sqrt{n} - 1\right) + \sqrt{n} \\
&= cn - \sqrt{n} + \sqrt{n} \\
&= cn
\end{aligned}$$

Vi har altså fundet at $P(n) = \Omega(n)$ for $c > 0$

Opgave 3

Vi har at: $P(n) = 2 \cdot P\left(n - 2\right) + n$, vi gætter at $P(n) = \Theta(2^{\frac{cn}{2}} - \frac{n}{2}$. Vi viser upper-bound via substitutions metoden:

$$\begin{aligned}
P(n) &= 2 \cdot P\left(n - 2\right) + n \\
&\leq 2 \cdot \left(2^{\frac{cn-2}{2}} - \frac{n}{2}\right) + n \\
&= 2^{\frac{cn-2}{2}+1} - n + n \\
&= 2^{\frac{cn}{2}}
\end{aligned}$$

Vi har altsp fundet at $P(n) = O(n)$ for $c > 0$.
På samme måde finder vi lower-bound:

$$\begin{aligned} P(n) &= 2 \cdot P\left(n-2\right) + n \\ &\geq 2 \cdot \left(2^{\frac{cn-2}{2}} - \frac{n}{2}\right) + n \\ &= 2^{\frac{cn-2}{2}+1} - n + n \\ &= 2^{\frac{cn}{2}} \end{aligned}$$

Vi har altså fundet at $P(n) = \Omega(n)$ for $c > 0$.

Task 3

Pseudocode for introsort [1]

Algorithm 1 Introsort

```
Introsort(A, i, j)
  Introsort_loop(A, i, j, 2 * Floor_log(j-i)
  Insertion_sort(A, i, j)

Introsort_loop(A, i, j, Counter)
while j-i >= 32 do
  if Counter = 0 then
    Heapsort(A, i, j)
    return
  end if
  Counter := Counter - 1
  p := Partition(A, i, j, Median_of_3(A[i], A[i+(j-i)/2], A[j-1]))
  Introsort_loop(A, p, j, Counter)
  j := p
end while
```

[1] http://www.cs.rpi.edu/~musser/gp/introsort.ps

4

Da Partitioneringen køre i $O(n)$ tid, og bliver kørt samme antal gange som dybden på rekursionstræet som er $lg(n)$, så må worst-case for introsort være $O(n \cdot lg(n))$

5

Heapsort er hurtig og effektiv. Den har værste tidsscenarie $\mathcal{O}(nlog(n))$, ligesom mergesort. Quicksort har værste tidsscenarie $\mathcal{O}(n^2)$. Heapsort foretrækkes på langsommere maskiner, da den kun skal bruge en konstant caching $\mathcal{O}(1)$ i modsætning til mergesort som bruger $\mathcal{O}(n)$.

6

Kørselstiden er godt nok $\mathcal{O}(n^2)$ for insertion sort i værste tilfælde. Til gengæld er den lineær altså, $\mathcal{O}(n)$ hvis den allerede er sorteret. Hvis den så næsten er sorteret vil det kun kræve få instruktioner, at sorterer den. Det er dog stadig en "langsom"algoritme, da den vil vokse lineært med n (i bedste tilfælde). Så hvis den er næsten sorteret men n er en googol ville andre alternativer måske være bedre.

Exam outline (Christian Enevoldsen)

1. **Divide**

   del problemet op i mindre delproblemer af samme type

2. **Conquor**

   løs delproblemet på en nem måde

3. **Combine**

   Samle alle delløsninger til et endeligt resultat

4. Master theorem til rekursive ligninger (tid)

   (a) **Form** $T(n) = aT(n/b) + f(n), a \geq 1, b > 1$

   (b) **Tilfælde 1** hvis $f(n) = \mathcal{O}(n^{log_b(a-\epsilon)}), \epsilon > 0 \rightarrow T(n) = \Theta(n^{log_b(a)})$

   (c) **Tilfælde 2** hvis $f(n) = \Theta(n^{log_b(a)}) \rightarrow T(n) = \Theta(n^{log_b(a)} lg(n))$

   (d) **Tilfælde 3** hvis $\Omega(n^{log_b(a+\epsilon)}), \epsilon > 0$ og $af(n/b) \leq cf(n), c < 1$ så er $T(n) = \Theta(f(n))$

5. **Mergesort forklaring**

   Sæt at man har et rum fyldt af mennesker. Vi vil gerne sorterer dem efter højde ved brug af merge sortering. Alle stiller sig på en række og vi deler hermed rækken op i 2 rækker (venstre og højre). Samme procedure gentages på venstre række indtil alle i den originale venstre række står alene. Nu tager vi 2 grupper (hvilke der kun er en i lige nu) og sætter dem sammen, ved at sorterer efter højden. Samme procedurer gentages indtil vi kun har en gruppe tilbage. Nu er den originale venstre side sorteret. Samme sker nu for højre side og vi ender med at have 2 grupper som sættes sammen som i de små delopgaver.

Exam subject outline - Nicklas Jacobsen qmr656

Divide: Split problemet i mindre problmer
Conquer: Når problemerne er små nok, så løs dem på en triviel måde
Combine: Kombinerer løsningerne til en stor løsning på det samlede problem

Substitutions metode

Metoden består af 2 step:
1: Gæt en løsning 2: Matematisk induktions til bevis at løsningen er rigtig.

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

Vi gætter $T(n) = O(n \cdot lg(n))$ Vi substituerer ind:

$$\begin{aligned} T(n) &\leq 2(c(n/2)lg(n/2)) + n \\ &\leq cn \cdot lg(n/2) + n \\ &= cn \cdot lg(n) - cn \cdot lg(2) + n \\ &= cn \cdot lg(n) - cn + n \\ &\leq cn \cdot lg(n) \end{aligned}$$

Bemærk det kun gælder for $n > 1$

Master method

Hvis vi har en recurrence på følgende form:

$$T(n) = aT(n/b) + f(n)$$

Så har har $T(n)$ følgende asymptotiske grænser.

1: Hvis $f(n) = O(n^{log_b a - \epsilon})$ ved en konstant $\epsilon > 0$, så er det ensbetydende med at $T(n) = \Theta(n^{log_b a})$

2: Hvis $f(n) = \Theta(n^{log_b a})$, så $T(n) = \Theta(n^{log_b a} lg(n))$

3: hvis $f(n) = \Omega(n^{log_b a + \epsilon})$ for en konstant $\epsilon > 0$, og hvis $a \cdot f(n/b) \leq c \cdot f(n)$ for en konstant $c < 1$, så $T(n) = \Omega(f(n))$

Eksempel - Merge-sort

Merge-sort er en Divide-and-Conquer sorterings algoritme og foregår i $O(n \cdot lg(n)$ tid. Algoritmen deler listen af tal op rekursivt til mindre del-lister indtil del-listerne har ét element, og derved bliver anset som været sorteret. For hver del-liste samler (merger) algoritmen listerne til en større del-liste, ved linært at samligne elementerne i listerne. Når der kun er en del-liste tilbage, er listen sorteret.
Merge-sort kan deles op i to del-algoritmer: 1. Merge som er $O(n)$ og 2. Divide som er $O(lg(n)$. Og den rekursive form er derved $T(n) = 2(n/2) + n$.

Divide and Conquer - Simon Warg

Divide into smaller problems

You can use D&C whenever you have a problem that can be divided down to smaller size subproblems of the same kind as the main problem.

Conquer

In this stage, we are solving each subproblem.

Combine

After each subproblem has been solved, combine the subsolution until they have solved the main problem.

Recurrences

For a D&C problem, recurrence illustrates very well how you would for each iteration divide the problem, by calling the same function again with a new smaller sized problem, until you hit the base case where you start to solve and combine.

MergeSort

MergeSort is a good example of D&C how you for example could sort a set of numbers. Lets say you have n numbers of integers to be sorted. Begin

by dividing the set into two new sets of size n/2 and keep dividing it until you are left with two problems of size 1. Since now each subproblem is a singleton, we can sort them two and combine it into one sorted set. Further, this sorted set will be combined with another sorted set and they are merged by comparing their elements with eachother.

Exam outline - Robert S. Rasmussen - dfs207

Points

- Divide
- Conquer
- Combine
- Master method
- Substitution method

Problem Instance

A list of numbers from 8 to 1 and how merge sort handles these, also why it's the worst kind of list for merge sort.