AD - Assigment 1

pwn274, vxl334, npd457, kgt356

February 2022

Recurrence Analysis

Task 1

Since all functions are on the form

$$T(n) = a \cdot T(n/b) + f(n),$$

we can make use of the master method (theorem 4.1 CLRS), and we thereby have the following results:

I. $p(n) \in \Theta(n^3)$ (first rule)

II. $p(n) \in \Theta(n^3)$ (third rule)

III. $p(n) \in \Theta(n^{\log_9 10})$ (first rule)

Proof of I

Since $\log_2(8) = 3$ we have that

$$f(n) := n^2 \in \mathcal{O}(n^{\log_b a - \epsilon}) = \mathcal{O}(n^{3-\epsilon}),$$

which holds for all $\epsilon \leq 1$. By 4.1.1 we now have that

$$p(n) \in \Theta(n^3)$$

Proof of II

Note that $\log_4(8) = 3/2$ and we thereby clearly have that

$$f(n) := n^3 \in \Omega(n^{\log_4 8 + \epsilon}) = \Omega(n^{3/2 + \epsilon})$$

for $\epsilon \leq 3/2$. We also see that

$$8\left(\frac{n}{4}\right)^3 = 8\frac{n^3}{8^3} = \frac{n^3}{64} \leq cn$$

for all $c \in [1/64, 1)$ (as $c < 1$ by the theorem). And it thereby holds that

$$p(n) \in \Theta(n^3)$$

by theorem 4.1.3.

Proof of III

We wish to show that

$$f(n) := n \log n \in \mathcal{O}(n^a)$$

where $a > 1$. We make use of L'Hopitals rule (LH), which we can use on limits that would result in a "$\frac{\infty}{\infty}$" (or other indeterminate) expression(s):

$$\lim_{n\to\infty} \frac{n \log n}{n^a} \overset{LH+(*)}{=} \lim_{n\to\infty} \frac{\log n + 1}{a n^{a-1}} \overset{LH}{=} \lim_{n\to\infty} \frac{1/n}{a^2 n^{a-2}} \overset{(**)}{=} \frac{1}{a^2} \lim_{n\to\infty} \frac{1}{n^{a-1}} = 0$$

where $(*)$ is under the assumption that $\log e = 1$ (or in other words that we use the natural logarithm) – choosing another base will not make a significant difference on the proof – and $(**)$ is the fact that $\lim_{n\to\infty} cf(n) = c \lim_{n\to\infty} f(n)$[1], which holds for non indeterminate limits.

This shows that n^a with $a > 1$ will increase faster than $n \log n$ as $n \to \infty$; in other words that $n \log n \in \mathcal{O}(n^a)$ for $a > 1$.

We have that $\log_9 10 \approx 1.04 > 1$, and therefore it will also hold that

$$f(n) = n \log n \in \mathcal{O}(n^{\log_9 10 - \epsilon})$$

for all $\epsilon \leq \log_9 10 - 1 \approx 0.4$. By theorem 4.1.1 we now have that

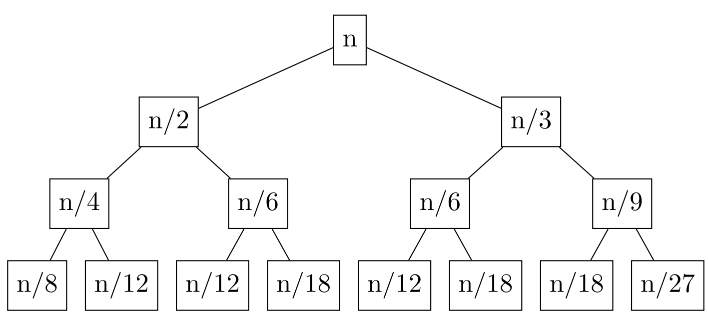
$$p(n) \in \Theta(n^{\log_9 10}).$$

Task 2

Starting from

$$p(n) = p(n/2) + p(n/3) + n \tag{1}$$

A qualified guess is $\mathcal{O}(n \log n)$, as the depth of the tree is $\log_2 n$ (left side). Since that depth upper-bounds all other depths, it will be the tightest upper-bound.



[1] See Section A

By a couple of calculations, it can easily be seen that the sum of row k totals $(5/6)^k \cdot n$.

The height of the tree is slightly more tricky, as the leftmost branch where we divide by 2 every step will converge after a depth of $\log_2(n)$, the rightmost branch will end after $\log_3(n)$ and the branches in between will have their final leafs somewhere in between. Using the worst case, the

$\sum_k^{\log_2(n)} (5/6)^k \cdot n \approx Cn$

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

Where the last is valid for $c \geq 6$.

Now starting from

$$p(n) = \sqrt{n} \cdot p(\sqrt{n}) + \sqrt{n} \tag{2}$$

Each level has $\prod_k \sqrt{n}^k$ branches of value $n^{1/2^k}$. This gives a total cost of:

$$\sum_i^d \prod_k^i n^{1/2^k} = \sum_i^d n^{\sum_k^i 1/2^k} = \sum_i^d n^{\sum_k^i 1/2^k} \leq \sum_i^d n = dn \tag{3}$$

Where d is the depth of the tree. Taking the hint I make a guess of $\mathcal{O}(n)$ by substituting with $cn - d$:

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

where the last inequality holds for all $d \geq 1$

Sorting

Task 3

```
introsort(A,i,j,c):
    if (c>2*log(len(A))):
        return Heapsort(A[i,j])
    else if (j-i < 16)
        return insertionsort(A[i,j])

    p = partition(A,i,j) #Skriv reference til hvor i bogen den er
    return introsort(A,i,p-1,c+1)
    return introsort(A,p+1,j,c+1)
```

Task 4

From the book we know that `heapsort` runs $\mathcal{O}(n \log n)$ and `quicksort` runs as $\mathcal{O}(n \log n)$. While `insertionsort` has an asymptotic, runtime of $\mathcal{O}(n^2)$ we have that it will always be a constant (as we only choose to use `insertionsort` if $j - i < c$). We thereby have that `insertionsort`s running time will be $\mathcal{O}(c^2) = \mathcal{O}(1)$ for any $c \in \mathbb{R}$ (here it $c \in \mathbb{N}$). Therefore, if $T(n)$ is the computing time of `introsort`, we have that

$$T(n) \in \mathcal{O}(n \log n).$$

Task 5

`heapsort` is used as it takes up less memory, and it will probably have less cache misses. If `mergesort` has run for quite some time, it will probably be taking up a lot of memory with several levels of recursion.

Task 6

Let $T_i(n)$ denote the computational complexity of `insertionsort` and $T_q(n)$ the same for `quicksort`. A fair approximation of T_i could be $T_i(n) \approx c_1 n^2$ for some constant $c_1 > 0$. Since $c_1 n^2 \in \Omega(n \log n)$ we have that

$$\Omega(n \log n) = \left\{ c_1 n^2 \mid \exists c_2 > 0, n_0 > 0 : n \geq n_0 \Rightarrow c_2 n \log n \leq c_1 n^2 \right\}. \tag{4}$$

`insertionsort` is quick on small arrays. We see that `quicksort` needs $\log n$ recursive calls to solve the array, which would take up memory and create a larger constant in front of $n \log n$ than what `insertionsort` has in front of n^2. This means that if

$$T_i(n) \leq c_1 n \log n \text{ and } T_q(n) \leq c_2 n^2$$

we have that for $n < n_0$ that $T_i(n) < T_q(n)$ (even though the asymptotic time complexity is higher for `insertionsort`) for some $c_1 << c_2$. Thereby, `insertionsort` can be faster than `quicksort` even though `insertionsort` has higher asymptotic complexity.

1 Individual parts

1.1 pwn274 - Jakob Hallundbæk Schauser

- Talk about name and core concept

  Recursion and solving sub-problems

- Explain merge-sort as an example
- Find running time by recursion tree

  $\mathcal{O}(n \log n)$

- Find running time by substitution method

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

- Find running time by master method

  (case 2)

1.2 vxl334 - Frederik Fabricius-Bjerre

Disposition for emnet 'Divide and Conquer algoritmer'

- Introducer del og hersk paradigmet
- Rekursionsligninger (vi bruger quicksort som udgangspunkt for fremlæggelsen)
- Opskriv rekursionsligning og dernæst rekursionstræ for quicksort

  Lav tidsanalyse med udgangspunkt i rekursionstræet: $\mathcal{O}(n \log n)$

- Introducer og opskriv master method sætningen

  Lav tidsanalyse med udgangspunkt i master method (case 2)

- Introducer substitutionsmetoden

  Lav tidsanalyse med udgangspunkt i substitutionsmetoden:

- Evt. hvis der er tid snak om nedre grænse for sortering ved del og hersk paradigmet.

1.3 npd457 - Sebastian Ø. Utecht

Del og Hersk disposition

- Generel definition af paradigmet
- Example: maximum-subarray problem
- Methods of solving:
  - Recursion Trees
  - Substitution method (combine with Recursion Trees)

- Master Method

1.4 kgt356 - Christoffer A. Ankerstjerne

Divide and Conquer

- Generel definition af Divide, Conquer, and combine
- Algoritmer eksempel: Quicksort, Mergesort
- Asymptotitsk køretid af Mergesort: $T(n) = \mathcal{O}(n \log n)$
  - Substitution method and recursion trees
  - Master Method

A Proof of mutiplitcation of limits

Let f and g be real or complex functions having the *finite* limits

$$\lim_{x \to x_0} f(x) = F \quad \textit{and} \quad \lim_{x \to x_0} g(x) = G \tag{5}$$

Then also the limit $\lim_{x \to x_0} f(x)g(x)$ exists and equals FG.
Proof:
Let ε be any positive number. The assumptions imply the existence of the positive numbers $\delta_1, \delta_2, \delta_3$ such that $|f(x) - F| < \frac{\varepsilon}{2(1+|G|)}$ when $0 < |x - x_0| < \delta_1$ (1)
$|g(x) - G| < \frac{\varepsilon}{2(1+|F|)}$ when $0 < |x - x_0| < \delta_2$, (2)
$|g(x) - G| < 1$ when $0 < |x - x_0| < \delta_3$.(3)

Note that (1), (2), and (3) can only hold if $F < \infty$ and $G < \infty$ (the limits are non indeterminate).

According to the condition (3) we see that $|g(x)| = |g(x) - G + G| \leqq |g(x) - G| + |G| < 1 + |G|$ when $0 < |x - x_0| < \delta_3$. Supposing then that $0 < |x - x_0| < \min\{\delta_1, \delta_2, \delta_3\}$ and using (1) and (2) we obtain

$$\begin{aligned}
|f(x)g(x) - FG| &= |f(x)g(x) - Fg(x) + Fg(x) - FG| \\
&\leqq |f(x)g(x) - Fg(x)| + |Fg(x) - FG| \\
&= |g(x)| \cdot |f(x) - F| + |F| \cdot |g(x) - G| \\
&< (1 + |G|)\frac{\varepsilon}{2(1 + |G|)} + (1 + |F|)\frac{\varepsilon}{2(1 + |F|)} \\
&= \varepsilon
\end{aligned}$$

This settles the proof.