Ugeopgave 1

Mathias Bramming Philip Meulengracht
Rasmus Borgsmidt

Task 1

I denne opgave bedes vi benytte mestermetoden (*master method*) til at angive øvre og nedre grænser for $p(n)$ i hvert af de følgende tilfælde.

$p(n) = 8p(n/2) + n^2$

Vi ser, at $T(n) = aT(n/b) + f(n) = 8p(n/2) + n^2$, hvor $a = 8, b = 2, f(n) = n^2$. Da vi har $f(n) = n^2 = O(n^{\log_b a - \epsilon}) = O(n^{\log_2(8) - \epsilon}) = O(n^2)$, hvor $\epsilon = 1$, er $T(n)$ ifølge mestermetoden asymptotisk begrænset med:

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^3).$$

$p(n) = 8p(n/4) + n^3$

Vi ser, at $T(n) = aT(n/b) + f(n) = 8p(n/4) + n^3$, hvor $a = 8, b = 4, f(n) = n^3$. Da vi har $f(n) = n^3 = \Omega(n^{\log_b a + \epsilon}) = \Omega(n^{\log_4(8) + \epsilon}) = \Omega(n^3)$, hvor $\epsilon = 3/2$, og da $af(n/b) = n^3/8 \leq cf(n) = n^3/2$, hvor $c = 1/2, n \geq 0$, er $T(n)$ ifølge mestermetoden asymptotisk begrænset med:

$$T(n) = \Theta(f(n)) = \Theta(n^3).$$

$p(n) = 10p(n/9) + n \log_2 n$

Ikke udført.

Task 2

I denne opgave bedes vi benytte indsætningsmetoden (*substitution method*) til at angive øvre og nedre grænser for $p(n)$ i hvert af de følgende tilfælde.

$p(n) = p(n/2) + p(n/3) + n$

Rekursionsligningen for $p(n)$ ligner meget (4.19) fra CLRS, så vi starter med følgende gæt:

$$p(n) = \Theta(n \lg n)$$

Vi ved fra opgaveteksten, at $p(1) = 1$ og $p(2) = 2$, så vi ser på tilfældet $n \geq 3$.

For den øvre grænse $p(n) = O(n \lg n)$ skal vi vise, at $p(n) \leq cn \lg n$ for en konstant $c > 0$ efter eget valg. Dette gøres ved induktion over n , hvor alle n rundes ned til nærmeste heltal.

Vi vælger basistilfælde $p(3) = p(1)+p(1)+3 = 5$ og $p(4) = p(2)+p(1)+4 = 7$. Med valg af f.eks. $c \geq 2$ opfyldes $p(n) \leq cn \lg n$ for $n \in \{3, 4\}$.

Vi antager herefter, at $p(m) \leq cm \lg m$ for alle $3 \leq m < n$; herunder specielt $p(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor$ og $p(\lfloor n/3 \rfloor) \leq c\lfloor n/3 \rfloor \lg \lfloor n/3 \rfloor$.

Induktionsskridtet vises således:

$$\begin{aligned}
p(n) &= p(\lfloor n/2 \rfloor) + p(\lfloor n/3 \rfloor) + n \\
&\leq c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor + c\lfloor n/3 \rfloor \lg \lfloor n/3 \rfloor + n \\
&\leq c(n/2) \lg(n/2) + c(n/3) \lg(n/3) + n \\
&= c(n/2) \lg n - c(n/2) \lg 2 + c(n/3) \lg n - c(n/3) \lg 3 + n \\
&= (5/6)cn \lg n - cn(1/2 + \lg(3)/3) + n \\
&\leq (5/6)cn \lg n - cn + n \\
&\leq cn \lg n
\end{aligned}$$

For den nedre grænse $p(n) = \Omega(n \lg n)$ skal vi vise, at $0 \leq cn \lg n \leq p(n)$ for en konstant $c > 0$ efter eget valg. Dette gøres ved induktion over n, hvor alle n rundes op til nærmeste heltal.

Vi vælger igen samme basistilfælde. Med valg af f.eks. $c = 1$ opfyldes $0 \leq cn \lg n \leq p(n)$ for $n \in \{3, 4\}$.

Vi antager herefter, at $0 \leq cm \lg m \leq p(m)$ for alle $3 \leq m < n$; herunder specielt $0 \leq c\lceil n/2 \rceil \lg \lceil n/2 \rceil \leq p(\lceil n/2 \rceil)$ og $0 \leq c\lceil n/3 \rceil \lg \lceil n/3 \rceil \leq p(\lceil n/3 \rceil)$.

Induktionsskridtet vises således:

$$\begin{aligned}
p(n) &= p(\lceil n/2 \rceil) + p(\lceil n/3 \rceil) + n \\
&\geq c\lceil n/2 \rceil \lg \lceil n/2 \rceil + c\lceil n/3 \rceil \lg \lceil n/3 \rceil + n \\
&\geq c(n/2) \lg(n/2) + c(n/3) \lg(n/3) + n \\
&= c(n/2) \lg n - c(n/2) \lg 2 + c(n/3) \lg n - c(n/3) \lg 3 + n \\
&= (5/6)cn \lg n - cn(1/2 + \lg(3)/3) + n \\
&\geq (5/6)cn \lg n, \quad && c < 1/2 + \lg(3)/3 = 1.0283 \\
&= c'n \lg n, && c' < (5/6) \cdot 1.0283 = 0.8569
\end{aligned}$$

Så ved at vælge f.eks. $c = 5/6$ opnås det ønskede resultat.

$$p(n) = \sqrt{n} + p(\sqrt{n}) + \sqrt{n}$$

Ikke udført.

$$p(n) = 2p(n - 2) + n$$

Ikke udført.

Task 3

We assume zero-based indexing in the following pseudo-code.

```
1  INTROSORT(A)
2    INTROSORTLOOP(A, 0, A.length, 2*lg(A.length), 16)
```

```
1  INTROSORTLOOP(A, p, r, maxdepth, threshold)
2    if (r - p) < threshold then
3      return INSERTIONSORT(A, p, r)
4    if maxdepth == 0 then
5      return HEAPSORT(A, p, r)
6    maxdepth = maxdepth - 1
7    pivot = MEDIAN(A, p, (p + r) / 2, r)
8    pi = PARTITION(A, p, r, pivot)
9    INTROSORTLOOP(A, p, pi-1, maxdepth, threshold)
10   INTROSORTLOOP(A, pi+1, r, maxdepth, threshold)
```

```
1  MEDIAN(A, p, q, r)
2    if A[q] < A[p] then
3      if A[r] < A[q] then
4        return A[q]
5      else
6        if A[r] < A[p] then
7          return A[r]
8        else
9          return A[p]
10   else
11     if A[r] < A[q] then
12       if A[r] < A[p] then
13         return A[p]
14       else
15         return A[r]
16     else
17       return A[q]
```

```
1  PARTITION(A, p, r, pivot)
2    i = p
3    exchange A[r] with A[pivot]
4    for p upto r
5      if A[p] <= A[r] then
6        exhange A[i] with A[p]
7        i = i + 1
8      p = p + 1
9    return i - 1
```

```
1  INSERTIONSORT(A, p, r)
2    for j = p + 1 to r
```

```
3       key  = A[j]
4       //insert A[j] into the sorted sequence A[0 .. j -1]
5       i = j - 1
6       while i >= 0 and A[i] > key
7         A[i + 1] = A[i]
8         i = i - 1
9       A[i + 1] = key
```

```
1  HEAPSORT(A, p, r)
2    BUILD-MAX-HEAP(A)
3    for r downto p
4      exchange A[p] with A[i]
5      p = p - 1
6      MAX-HEAPIFY(A, p, r)
```

Task 4

Introsort starts out as a regular quicksort, and if things go well switches to insertion sort when the remaining input is small and nearly sorted (a property of quicksort). If the data is distributed in a way that yields an excessive number of splits in the quicksort stage, the algorithm switches to heap sort with a worst-case running time of $n \log n$.

Because the algorithm switches to heap sort after no more than a constant factor of $\log n$ extraneous recursions, the combined worst-case running time is the same as that of heap sort, $n \log n$.

Task 5

Heap sort works in-place, whereas for instance merge sort uses n additional slots in the worst case (`http://en.wikipedia.org/wiki/Sorting_algorithm`).

Task 6

In the case that the array is sorted, only $n - 1$ comparisons have to be made, which gives insertion sort a best-case runtime of $O(n)$. Both the average and worst case of insertion sort are $O(n^2)$, but one of insertion-sorts properties is that it's adaptive, and thus performs significantly faster on already sorted input. Since introsort only switches to insertion sort when a certain level of deepness is reached, those cases can be avoided. Insertion sort works with low overhead, and is exceptional at sorting very small lists or nearly sorted lists.