Algorithms & Datastructures
Midterm Test 1

Wolfgang Pausch <wolfgang.pausch@uibk.ac.at>
Heiko Studt <heiko.studt@uibk.ac.at>
René Thiemann <rene.thiemann@uibk.ac.at>
Tomas Vitvar <tomas.vitvar@uibk.ac.at>

April 20th

Name	
Matrikelnummer	

Aufgabe	**mögliche Punkte**	**erreichte Punkte**
1.1	3	
1.2	6	
1.3	10	
2	13	
3.1	3	
3.2	8	
3.3	7	
Gesamt	**50**	

Exercise 1) Growth of functions **(in total 19 points)**

1. Write down the definition of $\Omega(f(n))$. (3 points)
2. Is $f(n) := 30 \cdot n^4 \in \Omega(n^3)$ true or not? Give a detailed proof, argueing with the definition of $\Omega(f(n))$. (6 points)
3. Consider the recurrence equations
   - $T_1(n) = 9 \cdot T(\frac{n}{9}) + n \cdot log_2(n^5)$
   - $T_2(n) = 25 \cdot T(\frac{n}{5}) + n \cdot log_2(n^5)$

   For $i \in \{1, 2\}$, either
   - find the solution $T_i(n) = \Theta(\dots)$ using the master theorem
   - or show why using the master theorem for T_i is not possible.

   Hint: You may assume that $n^\epsilon \notin \mathcal{O}(log(n))$ for all $\epsilon > 0$. (10 points)

Deutsch: Wachstum von Funktionen **(insgesamt 19 Punkte)**

1. Geben Sie die Definition der Menge $\Omega(f(n))$ an. (3 Punkte)
2. Gilt $f(n) := 30 \cdot n^4 \in \Omega(n^3)$? Geben Sie einen detailierten Beweis und argumentieren Sie mit der Definition von $\Omega(f(n))$. (6 Punkte)
3. Betrachten Sie die folgenden Rekursionsgleichungen:
   - $T_1(n) = 9 \cdot T(\frac{n}{9}) + n \cdot log_2(n^5)$
   - $T_2(n) = 25 \cdot T(\frac{n}{5}) + n \cdot log_2(n^5)$

   Für jedes $i \in \{1, 2\}$:
   - geben Sie die Lösung $T_i(n) = \Theta(\dots)$ mit Hilfe des Master-Theorems an,
   - oder begründen Sie, warum das Master-Theorem nicht für T_i anwendbar ist.

   Hinweis: Sie dürfen benutzen, dass $n^\epsilon \notin \mathcal{O}(log(n))$ für alle $\epsilon > 0$ gilt. (10 Punkte)

Exercise 1) Solution

1. The definition of the Ω-notation:

$$\Omega(f(n)) := \{g(n) \mid \exists c \in \mathbb{R}, n_0 \in \mathbb{N} : c > 0, \forall n \geq n_0 : c * f(n) \leq g(n)\} \tag{1}$$

2. Of course, $f(n) := 30 * n^4 \in \Omega(n^3)$ is true. Proof:

   Choose $c = 1$ and $n_0 = 1$. Then for all $n \geq n_0$ we have $c * n^3 = 1 * n^3 \leq 30n * n^3 = 30 * n^4 = f(n)$.

3. For T_2, the master theorem can be used, for T_1 not.

   - Have a look on $T_1(n) = 9 * T(\frac{n}{9}) + n * log_2(n^5)$. Here, $a = 9$ and $b = 9$, thus $log_b a = 1$. $f(n) = 5 * n * log_2(n)$. Case 1 of the master theorem obviously cannot be applied, since $f(n) \notin \mathcal{O}(n^{1-\epsilon})$ for any $\epsilon > 0$. Similarily, $f(n) \notin \Theta(n^1)$, since $n * log_2(n) \notin \mathcal{O}(n)$.
     Thus, for having a chance to apply the master theorem using case 3, $f(n) \in \Omega(n^{1+\epsilon})$ for some $\epsilon > 0$ must be true. This means, we have to find a $c > 0$ and a $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ the following is true: $c * n^{1+\epsilon} \leq n * log(n)$. In other words, $c * n^{\epsilon} \leq log(n)$ would be required. As $n^{\epsilon} \notin \mathcal{O}(log(n))$, case 3 is not applicable either, so the master theorem cannot be applied for $T_1(n)$.
   - Have a look on $T_2(n) = 25 * T(\frac{n}{5}) + n * log_2(n^5) = 25 * T(\frac{n}{5}) + 5 * n * log_2(n)$ (by using logarithm rules). Here, $a = 25$, $b = 5$ and $f(n) = 5 * n * log_2(n) \in \mathcal{O}(n^{2-\epsilon})$ for some $\epsilon > 0$.
     So, case 1 of the master theorem can be applied, and $T_2(n) \in \Theta(n^2)$.

Exercise 2) Maximum-Sort (13 points) Maximum-Sort sorts an array $A[i]$, $i \in 1 \dots n$ by iterating reversly over it and in each step, fetching the maximum of the subsequent elements.

In detail, it visits in descending order each of the elements $A[i]$, $i \in 1 \dots n$ and exchanges it with the maximum of the elements between $A[0]$ and $A[i]$.

Example: Given the array $(43, 42, 44, 41)$, Maximum-Sort performs the following steps:

1. Exchange 44 and 41, afterwards the array is $(43, 42, 41, 44)$
2. Exchange 43 and 41, afterwards the array is $(41, 42, 43, 44)$
3. (Exchange 42 and 42)
4. (Exchange 41 and 41)

Implement Maximum-Sort using the following signature.

Maximum-Sort method

```
static void swap(int[] a, int i, int j) {
    int tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
}

static void maximumSort(int[] a) {
    // Provide implementation here...
}
```

Deutsch: Maximum-Sort (13 Punkte) Maximum-Sort sortiert ein Feld $A[i]$, $i \in 1 \dots n$ durch Rückwärts-Iteration über das Array, wobei in jedem Schritt das aktuelle Element mit dem Maximum der vorherigen Elemente vertauscht wird.

Im Detail passiert folgendes: man iteriert in absteigender Folge über alle Elemente $A[i]$, $i \in 1 \dots n$ und vertauscht $A[i]$ mit dem Maximum der Werte $A[0], \dots, A[i]$.

Beispiel: Das Feld $(43, 42, 44, 41)$ wird durch Maximum-Sort wie folgt sortiert:

1. Vertauschung von 44 und 41, danach ist das Feld $(43, 42, 41, 44)$
2. Vertauschung von 43 und 41, danach ist das Feld $(41, 42, 43, 44)$
3. (Vertauschung von 42 und 42)
4. (Vertauschung von 41 und 41)

Implementieren Sie Maximum-Sort mit der oben angegebenen Signatur.

Exercise 2) Solution

Maximum-Sort

```
public class MaximumSort {

static void swap(int[] a, int i, int j) {
    int tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
}

static void maximumSort(int[] a) {
    for (int i = a.length-1; i > 0; i--) {
        int indexOfMaximum = i;
        for (int j = i-1; j >= 0; j--) {
            if (a[j] > a[indexOfMaximum]) {
                indexOfMaximum = j;
            }
        }
        swap(a, i, indexOfMaximum);
    }
}


public static void main(String[] args) {
    int[] testArray = {3,2,4,1};
    maximumSort(testArray);
    for (int i = 0; i < testArray.length; i++) {
        System.out.println("testArray[" + n + "] = " + testArray[n]);
    }
}
}
```

Exercise 3) Heap-Sort **(in total 18 points)**

1. Write down the heap condition, i.e. when (in general) is some given array a a heap? Is the array $(23, 5, 7, 1, 2, 6, 0)$ a heap? Why or why not? (3 points)

2. Sort the sequence $1, 2, 3, 4, 5$ using Heap-Sort. Draw the heap after each run of both downHeap and swap. (8 points)

3. Let h be the height of a heap, i.e. the maximal level. For example, a heap consisting of just the root node has height $h = 0$. Let $max_{el}(h)$ be the maximal possible number of elements of a heap of height h. For example, a heap of height $h = 1$ has got at most $max_{el}(1) = 3$ elements. Hints:

   - You may use the results you already proved on some exercise sheet.

   How many elements $max_{el}(h)$ has a heap of height h have at most? Give the value $max_{el}(h)$ in form of a non-recursive function and prove your result using induction. (7 points)

Deutsch: Heap-Sort **(insgesamt 18 Punkte)**

1. Notieren Sie die Heap-Bedinungung, d.h. wann ist ein Feld a ein Heap? Ist das Feld $(23, 5, 7, 1, 2, 6, 0)$ ein Heap? Begründen Sie kurz? (3 Punkte)

2. Sortieren Sie die Sequenz $1, 2, 3, 4, 5$ mittels Heap-Sort. Zeichnen Sie den Heap nach jeder Ausführung von downHeap und swap. (8 Punkte)

3. Sei h die Höhe eines Heaps, d.h. die maximale Ebene. Zum Beispiel hat ein Heap, der nur aus der Wurzel besteht, die Höhe 0. Sei $max_{el}(h)$ die maximal mögliche Anzahl von Elementen, die ein Heap der Höhe h haben kann. Zum Beispiel kann ein Heap der Höhe $h = 1$ maximal $max_{el}(1) = 3$ Elemente haben. Hinweise:

   - Sie dürfen auch Resultate über Heaps verwenden, die bereits im Proseminar bewiesen wurden.

   Wie viele Elemente $max_{el}(h)$ kann ein Heap der Höhe h maximal haben? Geben Sie den Wert von $max_{el}(h)$ in Form einer nicht-rekursiven Funktion an und beweisen Sie Ihr Resultat mittels Induktion. (7 Punkte)

Exercise 3) Solution

1. The heap condition is: Given some array a of size n, the following must hold: $\forall 0 \leq i < n : a[i] \geq a[2i+1] \wedge a[i] \geq a[2i+2]$. The given sequence is a heap.
2. Execute downHeap for 2, get $1, 5, 3, 4, 3$. Execute downHeap for 1, get $5, 4, 3, 1, 2$ (two iterations). Now perform the following swap / downHeap steps:
   - Swap 2 and 5 and get $2, 4, 3, 1 \mid 5$. Call downHeap for the 2 and get $4, 2, 3, 1 \mid 5$.
   - Swap 1 and 4 and get $1, 2, 3 \mid 4, 5$. Call downHeap for the 1 and get $3, 2, 1 \mid 4, 5$.
   - Swap 1 and 3 and get $1, 2 \mid 3, 4, 5$. Call downHeap for the 1 and get $2, 1 \mid 3, 4, 5$.
   - Swap 1 and 2 and get $1 \mid 2, 3, 4, 5$. Call downHeap for the 1 and nothing changes.
3. A heap of height h has at most $max_{el}(h) = 2^{h+1} - 1$ elements. Proof:
   - $max_{el}(0) = 1 = 2^{0+1} - 1$
   - $h \rightarrow h+1$: $max_{el}(h+1) = max_{el}(h) + n(h+1)$, since the biggest possible heap of height $h+1$ is
     - the biggest possible heap of height h, plus
     - all elements of height $h+1$

     With induction assumption: $max_{el}(h+1) = max_{el}(h) + n(h+1) = (2^{h+1} - 1) + 2^{h+1} = 2 * 2^{h+1} - 1 = 2^{h+1+1} - 1$