DGA Ü1

1) a) $\frac{\sum_{a \in A} (a^2)}{|A|}$ Summe der Quadrate durch Anzahl Elemente

b) Behauptung: $m = \sum_{i=1}^{k} a_i$ beim k-ten Schleifendurchgang

vor erster Iteration ist $m = 0 = \sum_{i=1}^{0} (a_i^2)$

Aufrechterhaltung: wenn $m = \sum_{i=1}^{k-1} (a_i^2)$, dann wird durch einen Schleifendurchgang $m = \sum_{i=1}^{k-1} (a_i)^2 + a_k \cdot a_k$

$= \sum_{i=1}^{k} (a_i)^2$

Terminierung: nach $|A|$ Durchgängen ist

$m = \sum_{i=1}^{|A|} (a_i^2) = \sum_{a \in A} (a_i^2)$

daher gibt der Algorithmus $\frac{\sum_{a \in A} (a_i^2)}{|A|}$ zurück

Vergleich Schleifeninvariante - Beweis durch vollständige Induktion

DGA Ü1

1) a) $\frac{\sum_{a \in A}(a^2)}{|A|}$

b) z.B. $m \geq 0$ vor erster Iteration offenbar

Aufrechterhaltung durch quadrieren gesichert

Terminierung mit positiver Zahl

(Annahme: Elemente aus A sind reell)

2)

Algorithmus (L)	Kosten	Durchläufe
tmp := int[L.length + 1];	$n \cdot c_1$	1
for i=0 to L.length do	c_2	n
tmp[L[i]] = 1;	c_3	n
for i=0 to L.length do	c_4	n
if tmp[i] == 1 then	c_5	n
print(i);	c_6	$\leq n$

$\Rightarrow T(n) \leq n \cdot (c_1 + c_2 + c_3 + c_4 + c_6) \quad \Rightarrow \mathcal{O}(n)$

3)

Horner(P, x):	Kosten · Durchläufe
res := 0;	c_1
for i=P.length-1 to 0 do	$n \cdot c_2$
res += P[i];	$n \cdot c_3$
res *= x;	$n \cdot c_4$
res += P[0];	c_5
return res;	c_6

$\Rightarrow O(n)$

Normal(P, x):	Kosten · Durchläufe
res := 0;	c_1
for i=0 to P.length do	$n \cdot c_2$
res += P[i] * x ^ i;	$n(c_3 + n \cdot c_4)$
return res;	c_5

$\Rightarrow O(n^2)$

DGA Ü1

4.) a) Algorithmus(M, X):

```
m := M.cnt_rows;
n := M.cnt_cols;
i := 0;    j := 0;
while !M.row[i].contains(X) do
    i := i+1;
while !M.col[j].contains(X) do
    j := j+1;
return (i, j);
```

b) benötigt im Durchschnitt $\frac{m}{2}$ Fragen bis Zeile gefunden und $\frac{n}{2}$ Fragen bis Spalte gefunden

$\Rightarrow \frac{m+n}{2}$ Fragen im Durchschnitt insgesamt

5) a) Algorithmus (A):

Algorithmus	Kosten · Durchlauf
//find average to distinguish between min and max	
avg := 0;	c_1
for i=0 to A.length:	$n \cdot c_2$
avg += A[i];	$n \cdot c_3$
avg /= A.length;	c_4
//now min ≤ avg ≤ max holds	
j := 0;	c_5
for i=0 to A.length:	$n \cdot c_6$
if A[i] ≤ avg:	$n \cdot c_7$
A.swp(i, j);	$n \cdot c_8$
j++;	$n \cdot c_9$

$\Rightarrow O(n)$

b) Ja, zuerst Sortieren nach kleinsten und größeren Werten

Dann den unsortierten größeren Teil nach gleichem Prinzip sortieren.

DGA Ü1

6) a) 2 4 3 1 7 6 8 9 0 5

$\underbrace{2\ 4}\ \underbrace{3}\ \underbrace{1\ 7}\ \underbrace{6\ 8\ 9}\ \underbrace{0\ 5}$ Aufteilung in Runs

$\underbrace{2\ 3\ 4}\ \underbrace{1\ 7}\ \underbrace{6\ 8\ 9}\ \underbrace{0\ 5}$ Zusammenführen einzelner Runs in bel. Reihenfolge (Parallelisierung optimieren?)

$\underbrace{2\ 3\ 4}\ \underbrace{1\ 6\ 7\ 8\ 9}\ \underbrace{0\ 5}$

$\underbrace{1\ 2\ 3\ 4\ 6\ 7\ 8\ 9}\ \underbrace{0\ 5}$

$\underbrace{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}$

b) Best Case Analyse: Alles bereits sortiert

Nach aufteilung in Runs fertig $O(n)$

Worst Case Analyse: Runs nur 1 Element groß

z.B. umgekehrt sortiert

Laufzeit so wie normales Mergesort $O(n \cdot \log(n))$