Nachweise der quadratischen Laufzeit bei Bubble-Sort

Bisher haben wir uns ja eigentlich nur experimentell Gewissheit über die Laufzeit von Bubble-Sort verschafft.

Dabei ergaben sich die Ergebnisse der letzten Stunde.Mit der nun vorhandenen mathematischen Definition der O-Notation können wir diese quadratische Laufzeit auch nachweisen. Dazu schauen wir uns den Quelltext der Sortierroutine an.

```
 1     public String gibSortiermethode()
 2     {
 3         return "BubbleSort";
 4     }
 5
 6     public void sortieren()
 7     {
 8         zZugriffe = 0;
 9         // gibt den Bereich, in dem Verglichen werden soll an
10         // Bereich wird jede Runde um 1 verkleinert
11         for(int aussen=zZahlen.length-1;aussen>0;aussen--)
12         {
13           for(int innen=0;innen<aussen;innen++)  // zu vergleichender Index wird bestimmt
14           {
15             zZugriffe = zZugriffe + 2;
16             if(zZahlen[innen]>zZahlen[innen+1])  // Array wird an den zuvor bestimmten Stellen verglichen
17             {
18               tauschen(innen,innen+1);   // wenn Zahl link größer als Zahl rechts, dann werden sie getauscht
19             }
20           }
21         }
22     }
```

Interessant scheinen die Zeilen 16-20 zu sein, da hier die eigentliche Sortierarbeit aufgebracht werden muss. Diese Zeile wird bei n Zahlen

- beim ersten Sortierdurchlauf (n-1)-mal,
- beim zweiten Sortierdurchlauf (n-2)-mal,
- beim dritten Sortierdurchlauf (n-3)-mal
- ...
- beim letzten Sortierdurchlauf 1 mal

aufgerufen.

Insgesamt gibt es also (n-1) Sortierdurchläufe bei n zu sortierenden Zahlen (was auch die for-i-Schleife in Zeile 5 zeigt), so dass (n-1)+(n-2)+(n-3)+....+1 die Zeile 10 aufgerufen wird. Diese Summe der ersten (n-1) Zahlen ergibt aber $0{,}5*(n-1)*(n-2) = 0{,}5*n^2-1{,}5*n+1$.

Nach unseren Vorgaben müssen wir nun vom „worst case" ausgehen, also davon ausgehen, dass in den Zeilen 16-20 immer(!) getauscht werden muss. Der Vergleich kostet zwei Speicherzugriffe, Vertauschen sechs Zugriffe, so dass ein Aufruf der Zeilen 16-20 acht Zugriffe kostet.

Somit ergibt sich eine Laufzeit von $4n^2-12n+8$ und dies liegt in $O(n^2)$.

Aufgabe:

a) Führe einen Nachweis der Laufzeit für den Algorithmus „Minsort" (Quellecodes siehe unten) durch.

b) Führe einen Nachweis der Laufzeit für den Algorithmus „rekursiver BubbleSort" (Quellecodes siehe unten) durch.
Hinweis: Bilde den Ablauf des rekursiven Bubble-Sort-Algorithmus zunächst mit Hilfe des Schachtelmodells ab (siehe nächste Seite)

c) Kannst Du auch Minsort rekursiv implementieren? Was bedeutet dies für die Laufzeit?

```
1     public String gibSortiermethode()
2     {
3         return "MinSort";
4     }
5
6     public void sortieren()
7     {
8         zZugriffe = 0;
9         int hMinIndex = 0;
10        // Durchgehen des kompletten Arrays, um für die Stelle j die passende Zahl zu finden
11        for(int j=0;j<zZahlen.length;j++)
12        {
13            hMinIndex = j;
14            // Suchen der kleinsten Zahl hinter den bereits sortierten Zahlen
15            for(int i=j;i<zZahlen.length;i++)
16            {
17                zZugriffe = zZugriffe + 2;
18                // wenn die Zahl kleiner als das bisherige Minimum ist
19                if(zZahlen[i]<zZahlen[hMinIndex])
20                {
21                    // Speichern des Index des neuen Minimums
22                    hMinIndex=i;
23                }
24            }
25            tausche(j, hMinIndex); // das Minimum wird an die Stelle j gesetzt
26        }
27    }
```

```
1     public String gibSortiermethode()
2     {
3         return "BubbleSort-Rekursiv";
4     }
5
6     private void sortieren(int pBereich)
7     {
8       if (pBereich > 1) then
9       {
10        for (int j = 1; j < pBereich; j++)
11        {
12          zZugriffe = zZugriffe + 2;
13          if(zZahlen[j]>zZahlen[j+1])  // Array wird an den zuvor bestimmten Stellen verglichen
14          {
15            tauschen(j,j+1);   // wenn Zahl link größer als Zahl rechts, dann werden sie getauscht
16          }
17        }
18        sortieren(pBereich-1);
19    }
20
21    public void sortieren()
22    {
23        zZugriffe = 0;
24        this.sortieren(zZahlen.length-1);
25    }
```

zZahlen:

0	1	2	3	4	5
131	3	92	9	172	48

sortieren(   )

pBereich =

j	zZahlen						zZugriffe
	0	1	2	3	4	5	
0							

sortieren(   )

pBereich =

j	zZahlen						zZugriffe
	0	1	2	3	4	5	
0							

sortieren(   )

pBereich =

j	zZahlen						zZugriffe
	0	1	2	3	4	5	
0							

sortieren(   )

pBereich =

j	zZahlen						zZugriffe
	0	1	2	3	4	5	
0							

sortieren(   )

pBereich =

j	zZahlen						zZugriffe
	0	1	2	3	4	5	
0							