

Algorithmen und Datenstrukturen 1

Serie 3

1. a) Ich sortiere die gegebenen Folgen durch direktes Einfügen:

7	12	14	2	13	8	5	4	57	5
<u>7</u>	12	14	2	13	8	5	4	57	5
<u>7</u>	<u>12</u>	14	2	13	8	5	4	57	5
<u>7</u>	<u>12</u>	<u>14</u>	2	13	8	5	4	57	5
<u>2</u>	<u>7</u>	<u>12</u>	<u>14</u>	13	8	5	4	57	5
<u>2</u>	<u>7</u>	<u>12</u>	<u>13</u>	<u>14</u>	8	5	4	57	5
<u>2</u>	<u>7</u>	<u>8</u>	<u>12</u>	<u>13</u>	<u>14</u>	5	4	57	5
<u>2</u>	<u>5</u>	<u>7</u>	<u>8</u>	<u>12</u>	<u>13</u>	<u>14</u>	4	57	5
<u>2</u>	<u>4</u>	<u>5</u>	<u>7</u>	<u>8</u>	<u>12</u>	<u>13</u>	<u>14</u>	57	5
<u>2</u>	<u>4</u>	<u>5</u>	<u>7</u>	<u>8</u>	<u>12</u>	<u>13</u>	<u>14</u>	<u>57</u>	5
<u>2</u>	<u>4</u>	<u>5</u>	<u>5</u>	<u>7</u>	<u>8</u>	<u>12</u>	<u>13</u>	<u>14</u>	<u>57</u>

4	6	3	14	15	3	7	9	12	1	6	2
<u>4</u>	6	3	14	15	3	7	9	12	1	6	2
<u>4</u>	<u>6</u>	3	14	15	3	7	9	12	1	6	2
<u>3</u>	<u>4</u>	<u>6</u>	14	15	3	7	9	12	1	6	2
<u>3</u>	<u>4</u>	<u>6</u>	<u>14</u>	15	3	7	9	12	1	6	2
<u>3</u>	<u>4</u>	<u>6</u>	<u>14</u>	<u>15</u>	3	7	9	12	1	6	2
<u>3</u>	<u>3</u>	<u>4</u>	<u>6</u>	<u>14</u>	<u>15</u>	7	9	12	1	6	2
<u>3</u>	<u>3</u>	<u>4</u>	<u>6</u>	<u>7</u>	<u>14</u>	<u>15</u>	9	12	1	6	2
<u>3</u>	<u>3</u>	<u>4</u>	<u>6</u>	<u>7</u>	<u>9</u>	<u>14</u>	<u>15</u>	12	1	6	2
<u>3</u>	<u>3</u>	<u>4</u>	<u>6</u>	<u>7</u>	<u>9</u>	<u>12</u>	<u>14</u>	<u>15</u>	1	6	2
<u>1</u>	<u>3</u>	<u>3</u>	<u>4</u>	<u>6</u>	<u>7</u>	<u>9</u>	<u>12</u>	<u>14</u>	<u>15</u>	6	2
<u>1</u>	<u>3</u>	<u>3</u>	<u>4</u>	<u>6</u>	<u>6</u>	<u>7</u>	<u>9</u>	<u>12</u>	<u>14</u>	<u>15</u>	2
<u>1</u>	<u>2</u>	<u>3</u>	<u>3</u>	<u>4</u>	<u>6</u>	<u>6</u>	<u>7</u>	<u>9</u>	<u>12</u>	<u>14</u>	<u>15</u>

b) Der korrigierte Sourcecode¹:

```
public class Sort
{
    void insertSort(Comparable[] F)
    {
        for ( int i = 1 ; i < F.length ; i++ )
        {
            Comparable e = F[i];
            int pos = correctPos(F, e);
            if ( i != pos )
            {
                moveRight(F, i, pos);
                F[pos] = e;
            }
        }
    }
}
```

¹um eine kleinere Verbesserung einführen zu können ($i = 1$), gehe ich davon aus das $n \geq 1$ gilt (Sortierung eines einzelnen Elementes macht ja keinen Sinn).

```

    }
}

int correctPos(Comparable[] F, Comparable elem)
{
    int i = 0;
    while ( F[i].compareTo(elem) < 0 ) i++;
    return i;
}

void moveRight(Comparable[] F, int from, int to)
{
    for ( int i = from; i < to; i++ )
    {
        F[i] = F[i+1];
    }
}
}

```

c) i. Vergleichen:

best-case : $n-2$, tritt bei einem umgekehrt vorsortierten Feld ein; für jedes Element muss genau ein Vergleich ausgeführt werden (außer für das erste und das letzte Element).

worst-case : $\sum_{i=1}^{n-2} i$ und somit $\frac{(n-1) \cdot (n-2)}{2}$, da bei einem vorsortierten Feld für jedes Element bis zum Ende der Liste Vergleiche ausgeführt werden müssen.

ii. Vertauschen:

best-case : 0, tritt bei einem vorsortierten Feld auf; kein Element muss usgetauscht werden.

worst-case : $\sum_{i=1}^{n-1} i$ und somit $\frac{(n-1) \cdot (n-2)}{2}$, da bei einem umgekehrt vorsortierten Feld für jedes Element an der i -ten Stelle $n-i$ Vertauschungen ausgeführt werden müssen.

d) Man könnte die Funktionen `correctPos` und `moveRight` zusammenfassen und die Schleife die die Vergleiche zum suchen der Position macht gleichzeitig zum Vertauschen der Werte verwenden. Gleichzeitig könnte die Schleife in `correctPos` vom momentanen Suchelement an beginnen und nicht immer vom Anfang der Liste:

```

Fuer i=2 bis n
    temp = A(i); j=i-1
    while j>0 and A(j)>temp
        A(j+1) = A(j); j--
    A(j+1) = temp

```

e) siehe c).

2. a) Bubblesort:

7	12	14	2	13	8	5	4	57	5
7	12	2	13	8	5	4	14	5	<u>57</u>
7	2	12	8	5	4	13	5	<u>14</u>	<u>57</u>
2	7	8	5	4	12	5	<u>13</u>	<u>14</u>	<u>57</u>
2	7	5	4	8	5	<u>12</u>	<u>13</u>	<u>14</u>	<u>57</u>
2	5	4	7	5	<u>8</u>	<u>12</u>	<u>13</u>	<u>14</u>	<u>57</u>
2	4	5	5	<u>7</u>	<u>8</u>	<u>12</u>	<u>13</u>	<u>14</u>	<u>57</u>

4	6	3	14	15	3	7	9	12	1	6	2
4	3	6	14	3	7	9	12	1	6	2	<u>15</u>
3	4	6	3	7	9	12	1	6	2	<u>14</u>	<u>15</u>
3	4	3	6	7	9	1	6	2	<u>12</u>	<u>14</u>	<u>15</u>
3	3	4	6	7	1	6	2	<u>9</u>	<u>12</u>	<u>14</u>	<u>15</u>
3	3	4	6	1	6	2	<u>7</u>	<u>9</u>	<u>12</u>	<u>14</u>	<u>15</u>
3	3	4	1	6	2	<u>6</u>	<u>7</u>	<u>9</u>	<u>12</u>	<u>14</u>	<u>15</u>
3	3	1	4	2	<u>6</u>	<u>6</u>	<u>7</u>	<u>9</u>	<u>12</u>	<u>14</u>	<u>15</u>
3	1	3	2	<u>4</u>	<u>6</u>	<u>6</u>	<u>7</u>	<u>9</u>	<u>12</u>	<u>14</u>	<u>15</u>
1	3	2	<u>3</u>	<u>4</u>	<u>6</u>	<u>6</u>	<u>7</u>	<u>9</u>	<u>12</u>	<u>14</u>	<u>15</u>
1	2	<u>3</u>	<u>3</u>	<u>4</u>	<u>6</u>	<u>6</u>	<u>7</u>	<u>9</u>	<u>12</u>	<u>14</u>	<u>15</u>

8	4	3	11	9	7	5	14	19	5	1	4
4	3	8	9	7	5	11	14	5	1	4	<u>19</u>
3	4	8	7	5	9	11	5	1	4	<u>14</u>	<u>19</u>
3	4	7	5	8	9	5	1	4	<u>11</u>	<u>14</u>	<u>19</u>
3	4	5	7	8	5	1	4	<u>9</u>	<u>11</u>	<u>14</u>	<u>19</u>
3	4	5	7	5	1	4	<u>8</u>	<u>9</u>	<u>11</u>	<u>14</u>	<u>19</u>
3	4	5	5	1	4	<u>7</u>	<u>8</u>	<u>9</u>	<u>11</u>	<u>14</u>	<u>19</u>
3	4	5	1	4	<u>5</u>	<u>7</u>	<u>8</u>	<u>9</u>	<u>11</u>	<u>14</u>	<u>19</u>
3	4	1	4	<u>5</u>	<u>5</u>	<u>7</u>	<u>8</u>	<u>9</u>	<u>11</u>	<u>14</u>	<u>19</u>
3	1	1	<u>4</u>	<u>5</u>	<u>5</u>	<u>7</u>	<u>8</u>	<u>9</u>	<u>11</u>	<u>14</u>	<u>19</u>
1	3	<u>4</u>	<u>4</u>	<u>5</u>	<u>5</u>	<u>7</u>	<u>8</u>	<u>9</u>	<u>11</u>	<u>14</u>	<u>19</u>

b) i. Vergleichen:

best-case : n , Liste ist schon vorsortiert.

worst-case : n^2 , Liste ist umgekehrt sortiert.

ii. Vertauschen:

best-case : 0, Liste ist schon vorsortiert.

worst-case : n^2 , Liste ist umgekehrt sortiert.

3. a) Shellsort:

$t = \lceil \log_2(10) \rceil = 3$, somit $h_1 = 7, h_2 = 3$ und $h_3 = 1$:

7	12	14	2	13	8	5	4	57	5
4 ₁	12 ₂	5 ₃	2 ₄	13 ₅	8 ₆	5 ₇	7 ₁	57 ₂	14 ₃
2 ₁	7 ₂	5 ₃	4 ₁	12 ₂	8 ₃	5 ₁	13 ₂	57 ₃	14 ₁
2 ₁	4 ₁	5 ₁	5 ₁	7 ₁	8 ₁	12 ₁	13 ₁	14 ₁	57 ₁

$t = \lceil \log_2(12) \rceil = 3$, somit $h_1 = 7, h_2 = 3$ und $h_3 = 1$:

4	6	3	14	15	3	7	9	12	1	6	2
4 ₁	6 ₂	1 ₃	6 ₄	2 ₅	3 ₆	7 ₇	9 ₁	12 ₂	3 ₃	14 ₄	15 ₅
3 ₁	2 ₂	1 ₃	4 ₁	6 ₂	3 ₃	6 ₁	9 ₂	12 ₃	7 ₁	14 ₂	15 ₃
1 ₁	2 ₁	3 ₁	3 ₁	4 ₁	6 ₁	6 ₁	7 ₁	9 ₁	12 ₁	14 ₁	15 ₁

$t = \lceil \log_2(12) \rceil = 3$, somit $h_1 = 7, h_2 = 3$ und $h_3 = 1$:

8	4	3	11	9	7	5	14	19	5	1	4
8 ₁	4 ₂	3 ₃	1 ₄	4 ₅	7 ₆	5 ₇	14 ₁	19 ₂	5 ₃	11 ₄	9 ₅
1 ₁	4 ₂	3 ₃	5 ₁	4 ₂	7 ₃	5 ₁	11 ₂	9 ₃	8 ₁	14 ₂	19 ₃
1 ₁	3 ₁	4 ₁	4 ₁	5 ₁	5 ₁	7 ₁	8 ₁	9 ₁	11 ₁	14 ₁	19 ₁

b) i. Vergleichen:

best-case : vorsortierte Liste, n ist abhängig von der gewählten Schrittweite.

worst-case : ebenfalls abhängig von der Schrittweite, mit kleinerer Schrittweite steigt die Anzahl von Vergleichen bis zu der beim Insertion Sort benötigten, umgekehrt vorsortierte Liste.

ii. Vertauschen:

best-case : $n = 0$, vorsortierte Liste.

worst-case : ebenfalls abhängig von der Schrittweite, umgekehrt vorsortierte Liste.