

Übungsblatt 9

Adrian Schollmeyer

Aufgabe 1

(a)

Split:

- 24, 17, 13, 42 | 9, 37, 1
- 24, 17 | 13, 42 | 9, 37 | 1
- **24 | 17 | 13 | 42 | 9 | 37 | 1**

Merge:

- **17, 24 | 13, 42 | 9, 37 | 1**
- **13, 17, 24, 42 | 1, 9, 37**
- 1, 9, 13, 17, 24, 37, 42

(b)

Pivot-Elemente sind fett markiert. Elemente, die vertauscht werden, sind unterstrichen.

```

24  17  13  42   9  37  1
24 17  13  42   9  37  1
24 17  13  1   9  37  42
  9  17  13  1   24  37  42
  9  17 13  1   24  37  42
  9  1 13  17   24  37  42
  1   9  13 17   24  37  42
  1   9  13 17   24  37  42

```

Aufgabe 2

(a)

Listing 1: RandomQuicksortMain.java

```

1 package me.adrian.aup9;
2
3 /**
4  * Randomized QuickSort
5  * <p>

```

```
6 * begin and end iterators are C++-STL-Style, meaning the begin iterator points
  to the first element of the selected
7 * range and the end iterator points to one element after the selected range.
8 */
9 public class RandomQuicksortMain
10 {
11     private static void swap(int[] data, int iteratorA, int iteratorB)
12     {
13         int valAtA = data[iteratorA];
14         data[iteratorA] = data[iteratorB];
15         data[iteratorB] = valAtA;
16     }
17
18     public static int[] partition(int[] data, int iteratorBegin, int iteratorEnd
19         )
20     {
21         int nComparisons = 0;
22         int iteratorPivot = iteratorBegin + (int) (Math.random() * (iteratorEnd
23             - iteratorBegin));
24         int valuePivot = data[iteratorPivot];
25         int iteratorL = iteratorBegin;
26         int iteratorR = iteratorEnd - 1;
27         while (iteratorL < iteratorR) {
28             nComparisons++;
29             while (iteratorL < iteratorEnd && data[iteratorL] <= valuePivot)
30                 iteratorL++;
31             nComparisons++;
32             while (iteratorR >= iteratorBegin && data[iteratorR] > valuePivot)
33                 iteratorR--;
34
35             if (iteratorL < iteratorR) {
36                 swap(data, iteratorL, iteratorR);
37
38                 // Follow the position of the pivot element to put it back in
39                 // place later on
40                 if (iteratorL == iteratorPivot)
41                     iteratorPivot = iteratorR;
42
43                 if (iteratorR == iteratorPivot)
44                     iteratorPivot = iteratorL;
45             }
46             swap(data, iteratorPivot, iteratorR);
47         }
48
49     /**
50      * Sort the input using QuickSort
```

```

51     *
52     * @param data The input array
53     * @return The total amount of comparisons to the pivot element
54     */
55     public static int sort(int[] data)
56     {
57         return sort(data, 0, data.length);
58     }
59
60 /**
61 * Sort a specific range of the input array using QuickSort
62 *
63 * @param data      The input array
64 * @param iteratorBegin begin iterator of the selected range
65 * @param iteratorEnd end iterator of the selected range
66 * @return The total amount of comparisons to the pivot element
67 */
68     public static int sort(int[] data, int iteratorBegin, int iteratorEnd)
69     {
70         if (iteratorBegin < iteratorEnd - 1) {
71             int[] partRes = partition(data, iteratorBegin, iteratorEnd);
72             sort(data, iteratorBegin, partRes[0]);
73             sort(data, partRes[0] + 1, iteratorEnd);
74             return partRes[1];
75         }
76         return 0;
77     }
78 }
```

(b)

Der Test wurde mithilfe von JUnit geschrieben. Mit `@Test` annotierte Funktionen sind die auszuführenden Tests. Die Funktion `assertTrue(boolean)` prüft, ob die Eingabe wahr ist und bricht den Test mit einem Fehler ab, sofern dies nicht der Fall ist.

Listing 2: RandomQuicksortMainTest.java

```

1 package me.adrian.aup9;
2
3 import org.junit.Test;
4
5 import static org.junit.Assert.assertTrue;
6
7 public class RandomQuicksortMainTest
8 {
9
10    private boolean isSorted(int[] arr)
11    {
12        for (int i = 0; i < arr.length - 1; i++) {
13            if (arr[i] > arr[i + 1])
14                return false;
15        }
16    }
17}
```

```
15         }
16
17     return true;
18 }
19
20 @Test
21 public void simpleTest()
22 {
23     int[] arr = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
24
25     RandomQuicksortMain.sort(arr);
26
27     assertTrue(isSorted(arr));
28 }
29
30 @Test
31 public void bigTest()
32 {
33     int minimum = Integer.MAX_VALUE;
34     int maximum = 0;
35     int sum = 0;
36     final int nIterations = 1000;
37
38     for (int i = 0; i < nIterations; i++) {
39         int[] a = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
40         int nComparisons = RandomQuicksortMain.sort(a);
41         assertTrue(isSorted(a));
42         if (minimum > nComparisons)
43             minimum = nComparisons;
44
45         if (maximum < nComparisons)
46             maximum = nComparisons;
47
48         sum += nComparisons;
49     }
50
51     float avg = sum * 1.f / nIterations;
52
53     System.out.println(
54         "Completed runtime check with\n\tmin = " + minimum + "\n\tmax = "
55         + maximum + "\n\tavg = " + avg
56         + "\ncomparisons for " + nIterations + " iterations.");
57 }
```

Aufgabe 3

(a)

```

1: function POW(b, e)
2:   if b = 0  $\wedge$  e = 0 then
3:     return -1;                                 $\triangleright$  Invalid input.
4:   else if e = 1  $\vee$  b = 0 then
5:     return b;
6:   else if b = 1  $\vee$  e = 0 then
7:     return 1;
8:   fi
9:   var partRes;
10:  partRes := pow(b, floor(e / 2));
11:  partRes := partRes * partRes;
12:  if e mod 2 = 1 then
13:    partRes := partRes * b;
14:  fi
15:  return partRes;
16: end function
```

(b)

$$T(n) = T\left(\frac{n}{2}\right) + 1 \quad (1)$$

$$T(1) = 0 \quad (2)$$

Asymptotische Abschätzung mittels Master-Theorem Fall 2:

$$f(n) = 1 \quad (3)$$

$$a = 1 \quad (4)$$

$$b = 2 \quad (5)$$

$$f(n) = \Theta\left(n^{\log_b a}\right) \quad (6)$$

$$= \Theta\left(n^{\log_2 1}\right) \quad (7)$$

$$= \Theta(n^0) \quad (8)$$

$$= \Theta(1) \quad \checkmark \quad (9)$$

$$\implies T(n) = \Theta\left(n^{\log_b a} \log n\right) \quad (10)$$

$$= \Theta\left(n^{\log_2 1} \log n\right) \quad (11)$$

$$= \Theta(1 \cdot \log n) \quad (12)$$

$$= \Theta(\log n) \quad (13)$$

Daher löst der Algorithmus die Formel mit $O(\log k)$ Multiplikationsoperationen. □

Aufgabe 4

Listing 3: ExactKnapsackMain.java

```
1 package me.adrian.aup9;
2
3 public class ExactKnapsackMain
4 {
5     public static boolean[] getIndizes(int[] s, int[][] p)
6     {
7         boolean[] res = new boolean[s.length];
8         for (int i = 0; i < res.length; i++) {
9             // Default initialization
10            res[i] = false;
11        }
12        int mass = p[0].length - 1;
13        // First column ( $j = 0$ ) can be skipped, as it's always 0
14        for (int j = mass; j > 0; j--) {
15            for (int i = 0; i < s.length; i++) {
16                if (p[i][j] == 1 && !res[i]) {
17                    res[i] = true;
18                    mass -= s[i];
19                    j -= s[i] - 1;
20                    break;
21                }
22            }
23        }
24
25        return res;
26    }
27
28    public static void main(String[] args)
29    {
30        int[][] m = { { 0, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
31                      -1, -1, -1 },
32                      { 0, -1, 0, 1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
33                      -1, -1, -1 },
34                      { 0, -1, 0, 0, -1, 0, -1, 1, 1, -1, 1, -1, -1, -1, -1, -1, -1 },
35                      { 0, -1, 0, 0, -1, 0, 1, 0, 0, 1, 0, 1, -1, 1, 1, -1, 1 } };
36        int[] s = { 2, 3, 5, 6 };
37
38        boolean[] res = getIndizes(s, m);
39        for (int i = 0; i < s.length; i++) {
40            System.out.println(s[i] + " gehört " + (res[i] ? "" : "nicht ") + "
zur Lösung.");
41        }
42    }
43}
```

41 }

Aufgabe 5

(a)

Induktionsanfang:

- Trivial für $k = 0, k = 1$
- Lösung für $k = 2$ siehe Aufgabenblatt Abbildung 1, die Einzelfliese liegt an einer Ecke

Induktionsvoraussetzung: Die Lösung existiert für $k = n - 1$.

Induktionsbehauptung: Die Lösung existiert für $k = n$.

Induktionsschritt:

- Die Lösung für $k = n - 1$ wird viermal so gelegt, dass die vier Einzelfliesen zusammen ein Quadrat bilden, d.h. die Lösung für $k = n - 1$ wird beim jedes mal so gelegt, dass die Ecke mit der Einzelfliese in die Mitte der auszulegenden Fläche zeigt.
- Von den vier Einzelfliesen in der Mitte können drei beliebige herausgenommen und durch eine L-förmige Fliese ersetzt werden. Somit verbleibt genau eine Einzelfliese in der auszulegenden Fläche und der Rest wurde mit L-förmigen Fliesen ausgelegt. **Das Viertel mit der Einzelfliese wird noch so um seinen Mittelpunkt gedreht, dass die Einzelfliese wieder in die äußere Ecke zeigt.** Dies stellt die Lösung für $k = n$ dar.

□

(b)

$$n_{Fl} \dots \text{Anzahl ausgelegter L-förmiger Fliesen} \quad (14)$$

Induktionsanfang:

$$k = 0 \implies n_{Fl} = 0 = \frac{1}{3} (2^{2 \cdot 0} - 1) = \frac{1}{3} \cdot 0 \quad \checkmark \quad (15)$$

$$k = 1 \implies n_{Fl} = 1 = \frac{1}{3} (2^{2 \cdot 1} - 1) = \frac{3}{3} \quad \checkmark \quad (16)$$

Induktionsvoraussetzung:

$$k = n - 1 \implies n_{Fl_1} = \frac{1}{3} (2^{2(n-1)} - 1) \quad (17)$$

Induktionsbehauptung:

$$k = n \implies n_{Fl_2} = \frac{1}{3} (2^{2n} - 1) \quad (18)$$

Induktionsschritt:

$$n_{Fl_2} = 4n_{Fl_1} + 1 \quad (19)$$

$$\stackrel{\text{IV}}{=} 4 \cdot \frac{1}{3} (2^{2n-1} - 1) + 1 \quad (20)$$

$$= \frac{1}{3} \cdot 4 (2^{2n-1} - 1) + 1 \quad (21)$$

$$= \frac{4}{3} \cdot (2^{2n-2} - 1) + 1 \quad (22)$$

$$= \frac{4}{3} \cdot 2^{2n-2} - \frac{4}{3} + 1 \quad (23)$$

$$= \frac{4}{3 \cdot 4} \cdot 2^{2n} - \frac{1}{3} - 1 + 1 \quad (24)$$

$$= \frac{1}{3} 2^{2n} - \frac{1}{3} \quad (25)$$

$$= \frac{1}{3} (2^{2n} - 1) \quad (26)$$

□