Algorithmen und Datenstrukturen
Musterlösung 3

Martin Avanzini <martin.avanzini@uibk.ac.at>
Thomas Bauereiß <thomas.bauereiss@uibk.ac.at>
Herbert Jordan <herbert@dps.uibk.ac.at>
René Thiemann <rene.thiemann@uibk.ac.at>

29. März, zur Besprechung am 5. April

Aufgabe 1) Quicksort

1. Pivot-Funktion $\ell + \lfloor \frac{1}{3} \cdot (r - \ell) \rfloor$:

   - minimal: $[0, 1, 2, 5, 4, 3, 6, 7, 8, 9]$
   - maximal: $[0, 3, 6, 9, 1, 4, 5, 2, 7, 8]$

2. Die Laufzeit mittels Pivot-Funktion, welche den Median des zu sortierenden Bereiches auswählt, is unabhängig von der Reihenfolge der Arrayelemente.

Aufgabe 2) Bucket-Sort und Radix-Sort

1. Dies kann wie folgt implementiert werden.

```
1  public class RadixInplaceSolution {
2
3      static int radixBits = 8;
4
5      static void radixSortInplace(int[] a) {
6      int mask = 0;
7      for (int i=0; i<radixBits; i++) {
8          mask = (mask << 1) | 1;
9      }
10     for (int shift=0; shift < 32; shift += radixBits) {
11         bucketSortInplace(a, mask, shift);
12     }
13     }
14
15
16     static int keyOf(int e, int shift, int mask) {
17     return (e >>> shift) & mask;
18     }
19
20     static int [] computeIndices(int[] a,int mask,int shift) {
21     int[] buckets = new int[mask+1];
22     int n = a.length;
23
24     // erst berechnen wir die groessen der einzelnen buckets
25     for (int i=0; i<n; i++) {
26         int key = keyOf(a[i], shift, mask);
27         buckets[key]++;
28     }
29     // nun berechnen wir die start-indices der einzelnen buckets
30     buckets[mask] = n - buckets[mask];
31     for (int j=mask-1; j >= 0; j--) {
32         buckets[j] = buckets[j+1] - buckets[j];
33     }
34     return buckets;
35     }
36
37     static void bucketSortInplace(int[] a,int mask,int shift) {
38     int[] nexts = computeIndices(a, mask, shift); // nexts[k] :=
           naechste freie stelle in bucket mit schluessel k
39     int n = a.length;
40     int cnt = 0;
41     int i = n-1;
42
43     // invarianten:
44     //  - a[i+1],..., a[n-1] ist sortiert
45     //  - fuer all k: a[b[k]],...,a[nexts[k]-1] wobei b=computeIndices
           (a, mask, shift) ist sortiert
46     //  - jedes element wird maximal einmal nach links getauscht
47     while (i>0) {
48         int key = keyOf(a[i], shift, mask);
49         int next = nexts[key];
50         while (next <= i && keyOf(a[next],shift,mask) == key) {next
               ++;}
51         if (next < i) { // element a[i] gehoert in bucket nach links
52         swap(a,i,next); cnt++;
53         nexts[key] = next + 1;
54         } else { // element a[i] im richtigen bucket
55         i--;
```

```
56              }
57          }
58          assert (cnt <= n);
59          }
60
61          static void swap(int[] a, int i, int j) {
62          int tmp = a[i];
63          a[i] = a[j];
64          a[j] = tmp;
65          }
66  }
```

2. Es kann gezeigt werden dass die (asymptotische) lineare Laufzeit Komplexität in der insitu-Variante beibehalten wird. Hierfür verwenden wir die Beobachtung dass die Bedingung $next < i$ aus Zeile 51 maximal n mal zutrifft.

   Wegen der verwendeten swap-Operation ist oben angeführter Algorithmus nicht stabil. Ebenfalls verliert der Algorithmus dadurch die Eigenschaft der Sequentialität.

3. Da unsere inplace-Version von Bucket-Sort nicht stabil ist führt iteratives Anwenden von Bucket-Sort in der Definition von Radix-Sort im Allgemeinen nicht zu einem sortierten Array.

Aufgabe 3) Quick-Select

1. Die mittlere Reihe stellt die Mediane der sortierten 5er-Blöcke dar. Sei v der Median dieser Mediane. Ohne Einschränkung der Allgemeinheit können wir annehmen dass alle Mediane links von v kleiner gleich, und rechts von v größer gleich v sind. (Diese Situation können wir immer durch Permutierung der 5er-Blöcke erreichen). Somit beinhaltet der blau markierte Block nur Elemente kleiner gleich v, der grüne Block nur Elemente größer gleich v. Somit sind die Hälfte der $\frac{3}{5} \cdot n$ Elemente kleiner gleich bzw. größer gleich v. Berücksichtigen wir die zusätzlichen $\leqslant 4$ Elemente für welche kein Median errechnet wurde erhalten wir die oben angeführte Schranke.

2. Dies kann wie folgt implementiert werden.

```java
public class QuickSelectSolution {

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

   // finded Position des minimalen Elementes
   static int minimum(int[] a) {
      int n = a.length;
      if (n > 0) {
         int min = a[0];
         int index = 0;
         for (int j=1; j<n; j++) {
            if (a[j] < min) {
               index = j;
               min = a[j];
            }
         }
         return index;
      } else {
         return -1;
      }
   }

   static void quickSelect(int[] a, int x) {
      if (x < 0 || x >= a.length) {
         throw new RuntimeException("no x-th element");
      }
      swap(a, 0, minimum(a));
      if (x > 0) {
         quickSelect(a, 1, a.length-1, x);
      }
   }

   static void quickSelect(int[] a, int l, int r, int x) {
      while (l < r) {
         int p = pivotSelect(a,l,r);
         int v = a[p]; // Partition: von hier bis ...
         int i = l - 1;
         int j = r;
         swap(a,p,r);
         while (true) {
            do i++; while (a[i] < v);
            do j--; while (a[j] > v);
            if (i >= j) {
```

```
48                  break;
49               }
50               swap(a,i,j);
51            }
52            swap(a,r,i); // ... hier gleich zu quickSort
53            if (i > x) {
54               r = i-1;
55            } else if (i == x) {
56               return;
57            } else {
58               l = i+1;
59            }
60         }
61      }
62
63      static int pivotSelect(int[] a, int l, int r) {
64         int n = (r - l + 1) / 5;
65         if (n == 0) {
66            return (l+1)/2 + r/2;
67         } else {
68            for (int k=0; k < n; k++) {
69               // sort a[l + k + 0*n], a[l + k + 1*n], ... , a[l + k + 4*
                     n]
70               // with insertion sort
71               for (int i=1; i<5; i++) {
72                  int j = l + k + i*n;
73                  int ai = a[j];
74                  while (j >= l + n) {
75                     if (a[j-n] > ai) {
76                        a[j] = a[j-n];
77                     } else {
78                        break;
79                     }
80                     j -= n;
81                  }
82                  a[j] = ai;
83               }
84            }
85            int middle = l + 2*n + n / 2;
86            quickSelect(a, l + 2*n, l + 3*n - 1, middle);
87            return middle;
88         }
89      }
90   }
```