Algorithms & Datastructures
Laboratory Exercise Sheet 4

Wolfgang Pausch <wolfgang.pausch@uibk.ac.at>
Heiko Studt <heiko.studt@uibk.ac.at>
René Thiemann <rene.thiemann@uibk.ac.at>
Tomas Vitvar <tomas.vitvar@uibk.ac.at>

April 14th, to be discussed on April 20th

Exercise 1) Quicksort

1. Use quicksort with pivot function $pivot(a, l, r) = r$ for sorting the sequence 13, 19, 9, 5, 3, 8, 7, 4, 21, 2, 6, 11 on a sheet of paper.
2. What happens if you omit the preprocessing step?

Exercise 1) Solution

1. Execute Quicksort as follows:

   - Preprocessing step: Swap 13 and 2, resulting in the sequence 2, 19, 9, 5, 3, 8, 7, 4, 21, 13, 6, 11.
   - Process quicksort(a, 1, 11), swapping 19 and 6. After that i points to the 21 and j to the 4, thus the exit condition is fulfilled. Finally swap 21 and 11, resulting in $2, 6, 9, 5, 3, 8, 7, 4 \mid 11 \mid 13, 19, 21$.
   - Process quicksort(a, 1, 7). Pivot element is 4. Swap 6 and 3. In the next step, $i = 2$ and $j = 0$. Thus the exit condition is fulfilled and we swap 9 and 4. Thus we get $3 \mid 4 \mid 5, 6, 8, 7, 9$, the recursive call for 3 does not need to do anything, and
   - Process quicksort(a, 3, 7). The pivot element here is the 9, thus no swapping is performed and we
   - Call quicksort(a, 3, 6). The pivot element now is 7, i ascends to the 8 and j descends to the 6, thus the exit condition is fulfilled, we swap 8 and 7 and the result is $5, 6 \mid 7 \mid 8$.
   - Another call quicksort(a, 3, 4) with pivot element 6 results in no swapping and another recursive call quicksort(a, 3, 3), which exits without further action.
   - Also, the call of quicksort(a, 6, 6) exits without further action.
   - Perform quicksort(a, 9, 11), pivot element is 21
   - Perform quicksort(a, 9, 10) on $13, 19$, pivot element is 19, no swapping necessary, finally call quicksort(a, 9, 9) and we are done.

   Result: $2, 3, 4, 5, 6, 7, 8, 9, 11, 13, 19, 21$.

2. If we omit the preprocessing step:

   - In general j may run out of the array on the left side. (index out of bounds), depending on the situation.
   - In this example: We do not swap 13 and 2 at the beginning of the algorithm. Thus, 13 stays at the left. After the first step we get $6, 2, 9, 5, 3, 8, 7, 4 \mid 11 \mid 19, 13, 21$. Consider the left branch: Its pivot element is the 4, thus the algorithm partitions the left part as follows: $3, 2 \mid 4 \mid 5, 6, 8, 7, 9$. Again consider the branch on the left: If you start quicksort on $3, 2$, the break condition for j is never true, since there is no element $a[i] < 2$ left of 2.

     Thus the algorithm will fail with an ArrayIndexOutOfBoundsException in this step.

Exercise 2) Loop invariant for Minimum Consider the minimum-algorithm implemented as follows

Minimum code

```
1   static int minimum(int[] a) {
2      if (a.length > 0) {
3         int m = a[0];
4         int i = 0;
5         int j = 1;
6         while (j < a.length) {
7            if (a[j] < m) {
8               i = j;
9               m = a[j];
10           }
11           j++;
12        }
13        return i;
14     } else {
15        throw new RuntimeException("minimum not defined");
16     }
17  }
```

Give a suitable loop invariant for the loop starting at line 6 and ending at line 12. Using the invariant it should be possible to conclude $a[i] = min(a[0], \ldots, a[a.length - 1])$ for non-empty arrays a, where i is the return value of the algorithm.

Exercise 2) Solution A suitable loop invariant is:

$$\phi \quad := \quad m = min(a[0], \ldots, a[j-1]) \quad \wedge \quad a[i] = m \quad \wedge \quad j \leq a.length$$

(When leaving the loop we know that ϕ and $j \not< a.length$ are satisfied. So, from the last condition of ϕ and $j \not< a.length$ we conclude $j = a.length$ and thus, using ϕ we achieve the desired result $a[i] = min(a[0], \ldots, a[a.length - 1])$.)

Exercise 3) Radix-Sort

1. Sort the sequence COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW and FOX using Radix-Sort lexicographically (like a telephone book) on a sheet of paper, one char column at a time. Write down the element order after each step of Radix-Sort.

2. Implement Radix-Sort, sorting radixBits at a time. Use the following skeleton code, which you can download:

Radix Sort

```
public class Sorting {

   static int radixBits = 8;

   static void radixSort(int[] a) {
      // Provide implementation here...
   }

   static void bucketSort(int[] a,int[] b,int mask,int shift) {
      int[] cnt_ind = new int[mask+1];
      int n = a.length;
      // count
      for (int i=0; i<n; i++) {
         int key = (a[i] >>> shift) & mask;
         cnt_ind[key]++;
      }
      // compute start-indices
      cnt_ind[mask] = n - cnt_ind[mask];
      for (int j=mask-1; j >= 0; j--) {
         cnt_ind[j] = cnt_ind[j+1] - cnt_ind[j];
      }
      // sort
      for (int i=0; i<n; i++) {
         int key = (a[i] >>> shift) & mask;
         int index = cnt_ind[key];
         b[index] = a[i];
         cnt_ind[key] = index+1;
      }
   }
}
```

Exercise 3) Solution

1. Situation:
   - Initial: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX
   - After first step: SEA, TEA, MOB, TAB, DOG, RUG, DIG, BIG, BAR, EAR, TAR, COW, ROW, NOW, BOX, FOX
   - After second step: TAB, BAR, EAR, TAR, SEA, TEA, DIG, BIG, MOB, DOG, COW, ROW, NOW, BOX, FOX, RUG
   - After third and final step: BAR, BIG, BOX, COW, DIG, DOG, EAR, FOX, MOB, NOW, ROW, RUG, SEA, TAB, TAR, TEA
2. E.g. implement it as follows:

Radix Sort Solution

```
public class Sorting {

   static int radixBits = 8;

   static void radixSort(int[] a) {
      int mask = 0;
      for (int i=0; i<radixBits; i++) {
         mask = (mask << 1) | 1;
      }
      int[] one = a;
      int[] two = new int[a.length];
      for (int shift=0; shift < 32; shift += radixBits) {
         bucketSort(one, two, mask, shift);
         int[] tmp = one; one = two; two = tmp;
      }
      if (one != a) {
         System.arraycopy(one, 0, a, 0, a.length);
      }
   }

   static void bucketSort(int[] a,int[] b,int mask,int shift) {
      int[] cnt_ind = new int[mask+1];
      int n = a.length;
      // count
      for (int i=0; i<n; i++) {
         int key = (a[i] >>> shift) & mask;
         cnt_ind[key]++;
      }
      // compute start-indices
      cnt_ind[mask] = n - cnt_ind[mask];
      for (int j=mask-1; j >= 0; j--) {
         cnt_ind[j] = cnt_ind[j+1] - cnt_ind[j];
      }
      // sort
      for (int i=0; i<n; i++) {
         int key = (a[i] >>> shift) & mask;
         int index = cnt_ind[key];
         b[index] = a[i];
         cnt_ind[key] = index+1;
      }
   }
}
```