EU1 - Extra Exercise 1

ID1018

December 4, 2018

Algorithms

The least integer — find the flaw

A problem: determine the least integer

A sequential collection of integers exist. Determine the least integer in the collection.

The solution

The solution below contains two errors, on different lines of code. The first error prevents the algorithm from halting. The second error produces the wrong answer for some integer sequences.

The errors in the solution are to be found and corrected. When correcting an error, only a portion of a single line of code is to be changed.

A solution to the problem — with two errors

```
// The min method returns the least element in a sequential
// collection. If the collection is empty an
// IllegalArgumentException is thrown.
public static int min (int[] elements)
    throws IllegalArgumentException
{
    if (elements.length == 0)
        throw new IllegalArgumentException ("empty collection");

    // Is used in trace printing 2:
    // int    nofIters = 1;

    int[]    sequence = elements;
    int      nofPairs = sequence.length / 2;
    int      nofUnpairedElements = sequence.length % 2;
```

```
    int      nofPossibleElements = nofPairs + nofUnpairedElements;
    int[]    partialSeq = new int[nofPossibleElements];
    int      i = 0;
    int      j = 0;
    while (sequence.length > 1)
    {
        // extract a partial sequence of possible elements
        i = 0;
        j = 0;
        while (j < nofPairs)
        {
            partialSeq[j++] = (sequence[i] < sequence[i + 1]) ?
                              sequence[i] : sequence[i + 1];
            i += 2;
        }
        if (nofUnpairedElements == 1)
            partialSeq[j] = sequence[sequence.length - 1];

        // now turn to the partial sequence
        sequence = partialSeq;
        nofPairs = nofPossibleElements / 2;
        nofUnpairedElements = nofPossibleElements % 2;
        nofPossibleElements = nofPairs + nofUnpairedElements;

        // Trace printing 1 - to follow the sequence
        // System.out.println (java.util.Arrays.toString (sequence));

        // Trace printing 2 - to terminate the loop preemptively
        // (to be able to see what happens initially)
        // if (nofIters++ == 10)
        //    System.exit (0);
    }

    // sequence[0] is the only remaining possible element
    // - it is the least element
    return sequence[0];
}
```

Exercises on the problem and solution

1. Trace the method `min` on a sequence of 16 elements. Use pen and paper; draw a series of pictures showing the transformation of the number sequence. Analyse transformations of the sequence and find the first error in the method. Correct the error.

2. To find the second error, trace the corrected version of the `min` method. Use a sequence of 19 elements with the least element in position 17. Draw a series of pictures that shows how the least element is determined. Why is there an erroneous answer? Find the other error in the method and correct it.

3. Use the computer to trace the original version (with two errors) of the

`min` method. Create a test program that calls the method `min`. Use a sequence of 16 elements. First activate trace printing 1 och follow what happens. Then also activate trace printing 2. Analyse the data you get and find the first error. Correct this error.

4. Use the computer to trace the version of the `min` method that only contains the second error. Use a sequence of 19 elements with the least element on position 17. Activate trace printing 1 only. Analyse the data you get, and find and correct the error.

5. Solve the problem in a different way; create a new method `min` that decides the least element in the collection. The new method does not use the separation strategy, but a more memory-efficient and simple strategy: the update strategy.