OTE: Ohjelmointitekniikka
Programming Techniques

course homepage: `http://www.cs.uku.fi/~mnykanen/OTE/`

Week 46/2008

Exercise 1. Complete checking claim 1 for the binary search algorithm in Figure 1 of the lecture notes by arguing the following informally:

(a) That claim 1 starts to hold after the initializations on line $1\frac{1}{2}$.

(b) That claim 1 continues to hold after line 6.

Exercise 2. Argue informally claim 2 as in the Exercise 1 above.

Exercise 3. The lectures and the two exercises above showed the correctness of the algorithm in question, if $l = u$ is true on line 7.

(a) Give a counterexample: an input where this is false.

(b) Propose a fix for this bug.

(c) Argue informally that the algorithm works after the fix.

Exercise 4. The Java listing on the next page shows an implementation for binary search, which is published in a Java textbook, impressive, well commented,... and incorrect.

(a) Let the input array `v` consist of two numbers `10` and `20`, and let the element `o` to find be `30`. What happens?

(b) Why does the listing attempt to handle the two-item case separately?

(c) How does the algorithm in Exercise 1 handle the two-item case?

Exercise 5. The listing on the next page attempts a *three-way* branching binary search: separate branches for the midpoint being less than, greater than or equal to the element sought.

(a) A correct three-way branching algorithm is hiding within the correctness argument for the two-way branching algorithm in Exercise 1. Where?

(b) Give an explicit pseudocode for this three-way branching version of the algorithm in Exercise 1.

(c) How would you argue for the correctness of your modified algorithm?

(d) Which of these two versions would you favour in practice? Why?

```
/**
 *  The statically accessible sort operation
 *
 *  @param v the sorted array of <code>Object</code>s to be
 *  searched.
 *
 *  @param o the object to be searched for.
 *
 *  @param c the <code>Comparator</code> used to compare the
 *  <code>Object</code> during the search process.  Must either be
 *  "less than" or "greater than" and the same comparator that
 *  defines the order on the array.
 *
 *  @return index of the item or -1 if it is not there.
 */
public static int execute(final Object[] v,
                          final Object o,
                          final Comparator c)
{
   int hi = v.length ;
   int lo = 0 ;
   while (true)
   {
        int centre = (hi + lo) / 2 ;
        if (centre == lo)
        {
           //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           return ( v[centre].equals(o)
                    ? centre
                    : ( v[centre+1].equals(o)
                      ? centre+1
                      : -1)) ;
        }
        if (c.relation(v[centre], o))
        {
           lo = centre ;
        }
        else if (c.relation(o, v[centre]))
        {
           hi = centre ;
        }
        else
        {
           return centre ;
        }
   }
}
```