Supervision work
- 20.10.2015 -

3.4 For the first function the compiler can see that it returns a data of the same type as the argument b, even though it doesn't know what type it is.

The second function calls itself, generating an infinite loop. In this case the compiler cannot relate the function's type with a type of an other parameter/ argument, so it just assumes that the data types of x and the function can be different.

4.3 The zip function does the opposite of the function from exercise 4.2. While that one splits a list into two different ones, the zip function joins two list into a list of pairs.

4.4 The "making chance" functions assume that *till* is a list of integers and *amt* is an integer. If this assumptions are violated, there will be an compiling error.

4.5 Let c_1 and c_2 be the values of the two coins.

So, $a * c_1 + b * c_2 = N$ and we have to find the number of the possible solutions a and b to satisfy this equation. Because both c_1 and c_2 are positive integer numbers, a can be any integer between 0 and $[N/c_1]$ and b will be determined in relation to it. In conclusion, there are $[N/c_1]+1$ or $O(N)$ ways to give change for N with 2 legal coins.

If we have K coins, then their values will be c_1, c_2, … c_K and the solutions will be a_1, a_2, … a_K.

So, $a_1 * c_1 + … + a_K * c_K = N$. a_i can be any value between 0 and $[N/c_i]$ => it can be chosen in $[N/c_i]$ +1 ways (for any i, $1 \leq i < K$). a_K will be determined from the values of the other solutions. Therefore, the number of ways we can give change for N with K legal coins is the product of $([N/c_i] + 1)$, with $1 \leq i < K$. In a simplified version, there are $O(N^{K-1})$ ways of giving change.

4.6 `> val f = fn : 'a * 'b -> 'b * 'a`

The argument of function f is a tuple, consisting of 2 elements, of two different, but unknown types: a and b. The function returns another tuple and, in this case, the first element is of type b and the second of type a.

If we take the case of f(1, true), then a = int and b = bool. So the function will return a tuple of the form bool * int.

```
> val g = fn : 'a -> 'a list
```

The g function has only one argument of type a and returns a list of elements of the same type.

So, for the example g 0, a = int and it will return a list of integers (int list).

5.1

For every element that needs to be inserted in the sorted list, the algorithm needs to go through the unsorted one in order to find the right number. So, for the 1st element it does n iterations to find it, for the 2nd, n-1 and so on.

Therefore, the algorithm does $1+2+3+…+n = n*(n-1)/2$ iterations. In conclusion, the time complexity of the algorithm will be $O(n^2)$.

5.3

After the algorithm swaps all the pairs in the list, the maximum number will be shifted to the end of the list and is fixed there. So no it has to sort a list with n-1 elements and so on until the entire list is sorted.

Therefore it needs to perform the swaps on the entire list n times, at the worst case and it needs n*n operations to do that. In conclusion, the complexity of the algorithm is $O(n^2)$.