Seminar #3

Strategy: problems should be solved pencil&paper based. All questions, analysis and tracings are assumed to be finalized BEFORE running the programs.

Objective: lists traversals, forward and backward approaches, nondeterministic calls

1. append3 – order, efficiency, nondeterministic calls.
2. delete – when/why we may/must have (i) a clause on the empty list; (ii) a !.
3. Implement a predicate to check whether a list provided as argument is sublist in another list which is also provided as argument. Examples:
    a. ?-sublist([b,c,d],[a,b,c,d,e]). – succeeds
    b. ?- sublist([b,c,d],[a,~~b~~,x,~~c~~,y,~~d~~,e]). – fails (although b,c,d occur in the larger list in the expected order, the first arg is NOT a sublist as the items do not occur contiguously).
    c. ?-sublist([b,c,d],[a,~~b~~,x,~~b,c~~,y,b,c,d,e]). –succeeds (although b is found WITHOUT being followed by c, the search continues, with the sublist being searched from the beginning).

    Compare the solutions (in terms of approach, efficiency)

4. Calculate the minimum element in a list:
    a. With forward recursion (and wrapper). Estimate big Oh function.
    b. With backward recursion following the implementation from forward recursion (with 2 recursive calls). Estimate big Oh function. Implementations a. and b. similar, yet different runtimes. Why? Explain and make a general interpretation for backward recursion.
    c. Backward improved (just one recursive call). Estimate big Oh function. ? Explain and make a general interpretation of when and how backward recursion can be improved.

    Trace (on paper) the execution of each, showing what is the partial result (the result at each specific order). Justify. Discuss the utility of partial results.

5. Generate the permutations of a list.
   Discussion: Prolog is just executable specification.
   The number of permutations is n!
   Start from the recurrence relation of n!, n!=n*(n-1) which is almost Prolog code.

```
n! =              % head of the clause, one perm of a set of size n
         n        %the way you can select the first item for the head of the output list
         (n-1)!   %recursive call on the input list without the selected item.
```

   Solutions:

   a. How many ways can you select all the elements in the list (one at a time). Think the nondeterministic way? Each way provides a different solution.
   b. How many ways can you find the list from which the selected element is missing?

c. From the solutions above mentioned, is something that can be omitted? If so, is there requirement for the rest of the predicates? Which and why?

Be aware: when the nondeterministic approach is employed for solving a problem, the predicate which should have the nondeterministic behavior needs to be cut-free (the implementation does NOT contain cut!).

Homework:

1. Given a list, split it in 2 disjoint lists of element < and respectively >= than a given element provided as argument (partition). To be used in quicksort.
2. Given two ordered lists, merge them to get one single ordered list of all elements. To be used in mergesort.
3. Given a list, make one traversal which swaps adjoin elements that are not in increasing order (swap). To be used in bubblesort.
4. Given a list, find whether an element is singleton (occurs just once in the list).