1. In the previous exercise you implemented a program to compute the Nth Fibonacci number:

```prolog
fib(1, 1).
fib(2, 1).
fib(N, F) :-
    N > 2,
    N2 is N-2, N1 is N-1,
    fib(N2, F2), fib(N1, F1),
    F is F1+F2.
```

For large values, this version takes too long. Use accumulators to implement a faster version. Why is the version with accumulators so much faster? You can test the running time of your predicates by using the built-in predicate `time`, for example, by running the following query:

```prolog
?- time(fib(20, X)).
```

2. In the previous exercise you also implemented a program to find the minimal element of a list:

```prolog
minElem([Min], Min).
minElem([Min|Tail], Min) :-
    minElem(Tail, TailMin),
    Min =< TailMin.
minElem([Head|Tail], TailMin) :-
    minElem(Tail, TailMin),
    Head > TailMin.
```

Implement the same predicate `minElem` which return the minimal element of a list using accumulators. For testing purposes, you can obtain a random list of length 20 from the following link: https://www.random.org/integer-sets/?sets=1&num=20&min=1&max=10000&commas=on&order=random&format=html. You can change the parameter `num=20` into some other numbers to obtain a list of that length.
3. The fictitious country of Elbonia issues stamps in the denominations of 15¢, 7¢, 3¢, and 1¢. Write a Prolog program that gets as input the total postage you have to pay and outputs how many stamps of each denomination you need to reach this total.

4. A directed graph can be represented in Prolog by listing the edges between nodes as facts. An edge from node a to node b would be represented by

   `edge(a,b).`

Define a predicate `path(S,T,P)` that returns `true` if there is a simple (acyclic) path from S to T. Otherwise it returns `false`. Hint: you can use the `member/2` predicate of Prolog as `member(X, L)` which returns true if the list L contains element X.

5. In the previous exercise, you implemented a predicate to compute Yth power of an integer X:

   `pow(_,0,1).
   pow(X,Y,Z) :- Y > 0, Y1 is Y-1,
                 pow(X,Y1,Z1),
                 Z is Z1*X.`

Implement an interactive Prolog program that compute the Yth power of X using the predicate `pow(X,Y,Z).` Hint: use the built-in unary predicate `integer` to check whether the inputs are integers. The program should stop when the user inputs `exit` or `quit`. An example of a running program:

```
?- pow.
Input number:
|: 2.
Input exponent:
|: 4.
16
Input number:
|: notnumber.
Please input an integer.
Input number:
|: quit.
true.
```

6. Implement a predicate `mathlist(Op, L1, L2, Res)` that applies binary operation `Op` between each pair of elements with the same index in L1 and L2 (i.e., first elements of L1 and L2, second elements of L1 and L2, etc.) resulting in the new list Res. Assume that L1 and L2 are always of the
same length. Test your implementation with the built-in mathematical predicates in Prolog such as `plus` and with your own predicates such as `pow`.

7. (a) Implement a generic mergesort algorithm `mergesort(L,S,OrderPred)` which is true when `S` is a sorted version of list `L` using the sort predicate `OrderPred`. For example, the following query:

```prolog
?- mergesort([2,3,1,7,2,6], S, '<').
```

returns `S = [1, 2, 2, 3, 6, 7]`, whereas the following query:

```prolog
?- mergesort([2,3,1,7,2,6], S, '>').
```

returns `S = [7, 6, 3, 2, 2, 1]`.

(b) Implement an ordering predicate `agename` which, when applied on the `mergesort` predicate, sorts a list of persons by their age in an ascending order, and, if two persons have the same age, they are sorted alphabetically. For example, consider a knowledge base of people as follows:

```prolog
age(john,29).
age(tom,13).
age(mary,26).
age(ann,9).
age(jerry,13).
age(andy,13).
```

then the following query:

```prolog
?- mergesort([john,mary,tom,ann,jerry,andy],S,agename).
```

returns `S = [ann, andy, jerry, tom, mary, john]`. 