Opgave 1

Her er et program med mulige løsninger:

```
myst([H|T] - [H|S], T - S).

% Kan bruges til at definere en rotate

rotate(N, L, A) :-
    append(L, V, LV),
    rot(N,LV - V,A - []).

rot(0, X, X).
rot(N, L-V, NL - NV) :-
    N > 0,
    myst( L - V, ML - MV),
    N1 is N - 1,
    rot(N1, ML - MV, NL - NV).

% Man kan ogsaa bruge append, men da append selv er O(N) bliver
% denne loesning O(N^2).

rotten_rotate(0 ,L, L).
rotten_rotate(N, [H|T], A) :-
    N > 0,
    append(T, [H], Onestep),
    N1 is N - 1,
    rotten_rotate(N1, Onestep, A).

% Endelig er der ogsaa muligheden at samle leddene op i en liste
% for saa til sidst at klistre dem paa i enden med append.

collect_rotate(N, L, A) :-
    collect_rot(N, V - V , L, A).

collect_rot(0, F - [], R, A) :-
    append(R, F, A).

collect_rot(N, S - [H|NT], [H|T], A) :-
    N > 0,
    N1 is N - 1,
    collect_rot(N1, S - NT, T, A).
```

Og her en kørsel af dem:

```
| ?-  myst([1,2,3,4] - [1,2], X).

X = [2,3,4]-[2]

yes
| ?- myst([1,2,3,4] - [1,2], X), myst(X,Y).

X = [2,3,4]-[2]
Y = [3,4]-[]

yes
| ?- myst(X - Y, [1,2] -Z).

X = [A,1,2]
Y = [A|Z]

yes
| ?- rotate(2, [a,b,c,d,e,f], X).

X = [c,d,e,f,a,b] ?

yes
| ?- rotate(2, Y, [a,b,c,d,e,f]).

Y = [e,f,a,b,c,d] ?
```

Opgave 2:

Her er en løsning lavet ved bogens algoritme (fra Appendix B):

```
 ?-  conj_form(all(X,all(Y,s(X,Y)->(~(m(X)#all(Z,(t(X,Z)&(~m(Z))))))))).

implout: all(_15,all(_16,~s(_15,_16)# ~ (m(_15)#all(_22,t(_15,_22)& ~m(_22)))))

negin: all(_15,all(_16,~s(_15,_16)# ~m(_15)&exists(_22,~t(_15,_22)#m(_22))))

skolem: all(_15,all(_16,~s(_15,_16)# ~m(_15)& ~t(_15,f3(_16,_15))#m(f3(_16,_15))))

univout: ~s(_15,_16)# ~m(_15)& ~t(_15,f3(_16,_15))#m(f3(_16,_15))

conjnf: (~s(_15,_16)# ~m(_15))& ~s(_15,_16)# ~t(_15,f3(_16,_15))#m(f3(_16,_15))

clausify: [cl([],[s(_15,_16),m(_15)]),cl([m(f3(_16,_15))],[s(_15,_16),t(_15,f3(_16,_15))]

:- s(_15,_16), m(_15).
m(f3(_16,_15)) :- s(_15,_16), t(_15,f3(_16,_15)).
```

Sp. 2.b:

```
| ?- q(X),p(X).

X = c ?

X = c ?     og fortsaetter saadan ad infinitum!!!!!!!

| ?- q(c),!,p(X).

X = a ?

X = b ?

X = b ?     og fortsaetter saadan ad infinitum!!!!!!!
```

Opgave 3

```
bin  :: Int -> [Int]
bin 0  = [1]
bin n  = [1] ++ [a+b| (a,b) <- zip prev (tail prev)] ++ [1]
            where prev = bin (n-1)

pascal 0 = do print [1]
              return [1]

pascal n = do prev <- pascal (n-1)
              now <- return ([1] ++ [a+b| (a,b) <- zip prev (tail prev)]++[1])
              print now
              return now
```

Og en kørsel:

```
Hugs session for:
Prelude.hs
bin.hs
Main> bin 4
[1,4,6,4,1]
Main> pascal 4
[1]
[1,1]
[1,2,1]
[1,3,3,1]
[1,4,6,4,1]

Main> :t pascal
pascal :: (Num a, Num b) => a -> IO [b]
Main>
```

Opgave 4

De opgivne funktioner har følgende signaturer:

```
sd     ::  (Eq a) => [a] -> [a] -> [a]
sd     =   foldl (flip (db (==)))

db     :: (a -> a -> Bool) -> a -> [a] -> [a]
db f x []       = []
db f x (y:ys)  = if f x y then ys else y : db f x ys
```

og er simple omskrivninger af definitionen af mængde-differens operatoren der findes i biblioteksmodulet List.hs:

```
infix 5 \\

(\\)                     :: (Eq a) => [a] -> [a] -> [a]
(\\) = foldl (flip delete)

delete                   :: (Eq a) => a -> [a] -> [a]
delete                    = deleteBy (==)

deleteBy                 :: (a -> a -> Bool) -> a -> [a] -> [a]
deleteBy eq x []          = []
deleteBy eq x (y:ys)      = if x `eq` y then ys else y : deleteBy eq x ys
```

Bevisskitse til sp. 4.c

Funktionen `(db (==)) :: a -> [a] -> [a]` er en funktion af 2 parametre. Lad `x :: a` og `zs :: [a]`, vi ønsker da at vise at $x \notin$`(db (==)) x zs :: [a]`. Dette er trivielt tilfældet såfrem `zs = []` ved den første regel for `db`. Ved induktion i længden af listen `zs` fås resultatet ved den anden regel for `db`.

Lad os indføre funktionen `fdb = flip (db (==)) :: [a] -> a -> [a]`, der er identisk med `db (==)`, bortset fra at den forventer sine argumenter i den omvendte rækkefølge. `foldl` har typesignaturen `(x -> y -> x) -> x -> [y] -> x`, og sammenholdes dette med typen for `fdb :: [a] -> a -> [a]` ses det at `foldl fdb :: [a] -> [a] -> [a]`

Kaldet `foldl fdb ys xs` folder funktionen `fdb` på listen `ys` sammen med samtlige enkelt-elementer fra listen `xs`, dvs. samtlige forekomster af elementer i `xs` fjernes i `ys`.