%positiveValues(L,LO):- “LO is the list of the positive values in the input list L”

positiveValues([], []).

positiveValues([X|L], [X|LO]) :- X>0, positiveValues(L, LO).

positiveValues([X|L], LO) :- X=<0, positiveValues(L, LO).

---------------------------------------------------------------

%permute(L,LO) :- LO is a list obtained as a permutation of the input list L

permute([ ],[ ]).

permute(L,[X|R]) :- omit(X,L,M), permute(M,R).

omit(H,[H|T],T).

omit(X,[H|L],[H|R]) :- omit(X,L,R).

-------------------------------------------------------------

%mergesort algorithm

split([],[],[]). %subproblem (a)

split([A],[A],[]).

split([A,B|T],[A|P1],[B|P2]) :- split(T,P1,P2).

merge([],L,L). %subproblem (c)

merge(L,[],L).

merge([X|L],[Y|L1],[X|LO]):-X<Y,merge(L,[Y|L1],LO).

merge([X|L],[Y|L1],[Y|LO]):-Y=<X,merge([X|L],L1,LO).

mergeSort([],[]). %subproblem (b)

mergeSort([A],[A]).

mergeSort([X,Y|L1],L2) :- split([X,Y|L1],P1,P2), mergeSort(P1,S1),

mergeSort(P2,S2), merge(S1,S2,L2).

-----------------------------------------------------------------

vectorialProduct([],[],0).

vectorialProduct([X|L],[Y|L1],P):- vectorialProduct(L,L1,P1), P is X\*Y+P1.

-------------------------------------------------------------------

reverse([],[]).

reverse([X|L], LO):- reverse(L,LR),append(LR,[X],LO).

-----------------------------------------------------------------

palindrome(L) :- reverse(L,L).

----------------------------------------------------------------

listOfpairList([],\_,[], []).

listOfpairList([[X,X]|L],X, [X|L1],[X|L2] ):- listOfpairList(L,X,L1,L2).

listOfpairList([[X,Y]|L],X, [Y|L1],L2 ):- X=\=Y, listOfpairList(L,X,L1,L2).

listOfpairList([[Y,X]|L],X, L1,[Y|L2] ):- X=\=Y, listOfpairList(L,X,L1,L2).

listOfpairList([[Y,Z]|L],X, L1,L2 ):- X=\=Y, X=\= Z, listOfpairList(L,X,L1,L2).

--------------------------------------------------------------------------  
fibonacci(0,0).

fibonacci(1,1).

fibonacci(N,S) :- N>1, N1 is N-1, N2 is N-2, fibonacci(N1,S1), fibonacci(N2, S2), S is S1+S2.

subsetList(L,L1):- “L is a sublist of L1, meaning that all elements in L appear in L1”

subsetList([],L).

subsetList([X|L],L1):- member(X,L1),subsetList(L,L1).

Write a predicate second(X,List) which checks if X is the second element of List.

second(X,[Y,X|L).

Write a (Prolog) program which takes in input two ordered lists of integers and returns an ordered

list of integers containing all elements in the original lists.

merge([], L, L).

merge(L, [], L).

merge([X|L1],[Y|L2],[X|LO]):-X<Y, merge(L1,[Y|L2],LO).

merge([X|L1],[Y|L2],[Y|LO]):-Y=<X, merge([X|L1],L2,LO).

partition(X, [], [], []).

partition(X,[Y|L],[Y|LO],L2) :- Y=<X, partition(X,L,LO,L2).

partition(X,[Y|L],L1,[Y|L2]) :- X<Y, partition(X,L,L1,L2).

quicksort([], []).

quicksort([X|L], LO) :- partition(X,L,L1,L2), quicksort(L1,LO1),quicksort(L2,LO2),

append(LO1,[X|LO2],LO).