```
------- INSERT AN ELEM ON EVERY POSITION OF A LIST -------
% insert(elem, l1l2...ln) =
% = {elem} U l1l2...ln
% = {l1} U insert(l2...ln, elem)

% insert(L:list, E: element, R: result list)
% (i,i,o)


insert(E,L,[E|L]).
insert(E,[H|T],[H|R]):-
   insert(E,T,R).

------- PERMUTATIONS -------
% perm(l1l2...ln) =
% = [], if n = 0
% = insert(l1, perm(l2...ln)), otherwise

% perm(L:list, R: result list)
% (i,o)

perm([],[]).
perm([H|T],R1):-
   perm(T,R),
   insert(H,R,R1).

------- ARRANGEMENTS -------
% arr(l1l2...ln, k) =
% = l1, if k = 1
% = arr(l1l2...ln, k), if k >= 1
% = insert(l1, arr(l2...ln, k - 1)), if k > 1

% arr(L:list, K:number, R:list)
% (i,i,o)

arr([H|_],1,[H]).
arr([_|T],K,R):-
   arr(T,K,R).
arr([H|T],K,R1):-
   K > 1,
   K1 is K - 1,
   arr(T,K1,R),
   insert(H,R,R1).

------- COMBINATIONS -------
% comb(l1l2...ln, k) =
% = l1, if k = 1 and n >= 1
% = comb(l2...ln, k), if k >= 1
% = {l1} U comb(l2...ln, k - 1), if k > 1

% comb(L:list, K:number, R:list)
% (i,i,o)

comb([E|_],1,[E]).
comb([_|T],K,R):-
   comb(T,K,R).
comb([H|T],K,[H|R]):-
   K > 1,
   K1 is K - 1,
   comb(T, K1, R).
```

```
-------SUBSETS-------
% subset(l1l2...ln) =
% = [], if n = 0
% = {l1} U subset(l2...ln), if n >= 1
% = subset(l2...ln), if n >= 1

% subset(L:list, R:result list)
% (i,o)

subset([],[]).
subset([H|T],[H|R]):-
   subset(T,R).
subset([_|T],R):-
   subset(T,R).

-------LENGTH FOR A LIST-------
% myLength(l1l2...ln) =
% = 0, if n = 0
% = 1 + myLength(l2...ln), otherwise

% myLength(L:list, R:number)
% (i,o)

myLength([],0).
myLength([_|T],R1):-
   myLength(T,R),
   R1 is R + 1.

------- CHECK IF LENGTH IS EVEN -------
% checkEven(l1l2...ln) =
% = true, if myLength(l1l2...ln) % 2  == 0
% = false, otherwise

% checkEven(L:list)
% (i)

checkEven(L):-
   myLength(L,N),
   N mod 2 =:= 0.

------- CHECK IF LENGTH IS ODD -------
% checkOdd(l1l2...ln) =
% = true, if myLength(l1l2...ln) % 2  == 1
% = false, otherwise

% checkOdd(L:list)
% (i)

checkOdd(L):-
   myLength(L,N),
   N mod 2 =:= 1.

------- COUNT EVEN NUMBERS -------
% countEven(l1l2...ln) =
% = 0, if n = 0
% = 1 + countEven(l2...ln), if l1 % 2 == 0
% = countEven(l2...ln), otherwise

% countEven(L:list, R:number)
% (i,o)

countEven([],0).
countEven([H|T],R1):-
   H mod 2 =:= 0,
   !,
   countEven(T,R),
   R1 is R + 1.
countEven([_|T],R):-
   countEven(T,R).

------- COUNT ODD NUMBERS -------
% countOdd(l1l2...ln) =
% = 0, if n = 0
% = 1 + countOdd(l2...ln), if l1 % 2 == 1
% = countOdd(l2...ln), otherwise

% countOdd(L:list, R:number)
% (i,o)

countOdd([],0).
countOdd([H|T],R1):-
   H mod 2 =:= 1,
   !,
   countOdd(T,R),
   R1 is R + 1.
countOdd([_|T],R):-
   countOdd(T,R).

------- CREATE A LIST FOR A GIVEN INTERVAL [A,B] -------
% createList(a, b) =
% = [], if a = b + 1
% = {a} U createList(a + 1, b), otherwise

% createList(A:number, B:number, R:list)
% (i,i,o)

createList(A,B,[]):-
   A =:= B + 1.
createList(A,B,[A|R]):-
   A1 is A + 1,
   createList(A1,B,R).

------- ABSOLUT DIFFERENCE -------
% absDiff(a,b) =
% = a - b, if a >= b
% = b - a, otherwise

absDiff(A,B,R):-
   A >= B,
   R is A - B.
absDiff(A,B,R):-
   A < B,
   R is B - A.

------- CHECK THE ABS DIFF FOR A GIVEN CONDITION (HERE IS <= 2) -------
% checkAbsDiff(l1l2...ln) =
% = true, if n = 2 and absDiff(l1,l2) <= 2
% = checkAbsDiff(l2...ln), if absDiff(l1,l2) <= 2
% = false, otherwise
% checkAbsDiff(L:list)
% (i)

checkAbsDiff([H1,H2]):-
    absDiff(H1,H2,R),
    R =< 2.
checkAbsDiff([H1,H2|T]):-
    absDiff(H1,H2,R),
    R =< 2,
    checkAbsDiff([H2|T]).

------- INSERT SORT -------
% insertFirst(l1l2...ln, elem) =
% = {elem} U l1l2...ln

% insertFirst(L:list, E:element, R:list)
% (i,i,o)

insertFirst(L,E,[E|L]).

% insert(l1l2...ln, elem) =
% = list(elem) , if n = 0
% = l1l2...ln , if l1 = elem
% = {elem} U l1l2...ln, if elem < l1
% = {l1} U insert(l2...ln, elem)

% insert(L:list, E:element, R:list)
% (i,i,o)

insert([],E,[E]).
insert([H|_],E,[H|_]):-
    H=:=E,
    !.
insert([H|T],E,R1):-
    E < H,
    !,
    insertFirst([H|T],E,R1).
insert([H|T],E,[H|R]):-
    insert(T,E,R).

% sortare(l1l2...ln) =
% = nil , if n = 0
% = insert(sortare(l2...ln), l1) , otherwise

% sortare(L:list, R:result)
% (i,o)

sortare([],[]).
sortare([H|T],R1):-
    sortare(T,R),
    insert(R,H,R1).

------- CHECK ARITHMETIC PROGRESSION -------
% progression(l1l2...ln) =
% = true, if n = 3 and l2 = (l1 + l2)/2
% = progression(l2...ln), if l2 = (l1 + l2)/2
% = false, otherwise

% progression(L:list)
% (i)

progression([H1,H2,H3]):- H2 =:= (H1 + H3) /2.
progression([H1,H2,H3|T]):-
   H2 =:= (H1 + H3) /2,
   progression([H2,H3|T]).

------- PRODUCT ELEMS OF A LIST -------
% productElems(l1l2...ln) =
% = 1, if n = 0
% = l1 * productElems(l2...ln), otherwise

% productElems(L:list, R:number)
% (i,o)

productElems([],1).
productElems([H|T],R1):-
   productElems(T,R),
   R1 is H*R.

------- SUM ELEMS OF A LIST -------
% computeSum(l1l2...ln) =
% = 0, if n = 0
% = l1 + computeSum(l2...ln), otherwise

% computeSum(L:list, R:number)
% (i,o)

computeSum([],0).
computeSum([H|T],R1):-
   computeSum(T,R),
   R1 is R + H.

------- APPEND FOR 2 LISTS -------
% myAppend(l1l2...ln, p1p2...pm) =
% = p1p2...pm, if n = 0
% = {l1} U myAppend(l2...ln, p1p2...pm), otherwise

% myAppend(L:list, P:list, R:list)
% (i,i,o)

myAppend([],P,P).
myAppend([H|T],P,[H|R]):-
   myAppend(T,P,R).

------- CHECK IF A LIST IS IN INCREASING ORDER -------
% checkIncreasing(l1l2...ln)
% = true, if n = 2 and l1 < l2
% = checkIncreasing(l2...ln), if l1 < l2
% = false, otherwise

% checkIncreasing(L:list)
% (i)

checkIncreasing([H1,H2]):-
   H1 < H2.
checkIncreasing([H1,H2|T]):-
   H1 < H2,
   checkIncreasing([H2|T]).
```