A.

CÂMPEAN CĂTĂLIN
ALEXANDRU

```
f([], -1).
f([H|T], S) :-
    f(T, S1),
    S1 > 0, !,
    S is S1 + H.
f([_|T], S) :-
    f(T, S1),
    S is S1.
```

→

```
fAux([H|_], S, S1) :-
    H > 0,
    S1 < H, !,
    S is H.
fAux(_, S, S1) :-
    S is S1.
```

now f will become:

```
f([], -1).
f([H|T], S) :-
    f(T, S1),
    fAux([H|T], S, S1).
```

we have an auxiliary function replacing the recursive call, therefore the 2 cases are treated inside the newly added function. Moreover, in the old one we'll just have one branch from the previous 2.

CÂMPEAN CĂTĂLIN
ALEXANDRU

2B

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

Math. model:

insert(E, $L_1 \dots L_m$) = $E \cup L_1 L_2 \dots L_m$
$E_1 \cup$ insert(E, L_2 ..

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

Math. model

$$an(l_1 l_2 \dots l_m, k) = \begin{cases} l_1 & \text{if } k=1 \\ an(l_2, \dots l_m, k) & \text{if } k>=1 \\ insert(l_1, an(l_2 \dots l_m, k-1)) & \text{if } k>1 \end{cases}$$

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

- an(L - list, k - nr. of elements, R - result list)
  an(i, i, o)

Math. model:

$$myLength(l_1 \dots l_n) = \begin{cases} 0, & \text{if } n=0 \\ 1 + myLength(l_2 \dots l_n), & \text{otherwise} \end{cases}$$

```
myLength([], 0).
myLength([_|T], LENGTH):-
    myLength(T, LENGTH2),
    LENGTH is LENGTH2 + 1.
```

- myLength(L - list, N - number)
  myLength (i, o)

Math model:

insertList(l_1, l_m, list) = list if m=0
$l_1 \cup$ list($l_2 \dots l_m$, list), otherwise

```
insertList([],L,L).
insertList([H|T],L,[H|R]):-
    insertList(T,L,R).
```

- insertList(L: list, L: list, R: result list)
  insertList(i, i, o)

Math model:

strictlyAsc(l_1, l_m) = True, m=0 || m=1
False, $l_1 >= l_2$
strictlyAsc($l_2 \dots l_m$), $l_2 > l_1$

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

- strictlyAsc(L: list)
- strictlyAsc(i)

```
oneSol(L,K,ARR):-
    an(L,K,ARR),
    strictlyAsc(ARR).
```

- oneSol(L: list, K: number, ARR: result list)
  oneSol(i, i, o)

```
allSol(_,1,_).
allSol(L,K,RR):-
    findall(RPartial, oneSol(L,K,RPartial),R1),
    K1 is K-1,
    allSol(L,K1,K2),
    insertList(R2,R1,RR).
```

```
wrapper(L, R):-
    myLength(L, LENGTH),
    allSol(L, LENGTH, R).
```

- call wrapper([1, 8, 6, 4], R])

- we use arrangements as in the given ex we can have [4,8] which wouldn't have happend if we chose combinations
- we make allSol recursive so that we can call it for every K from the length of the list to 2 (1 is the stop case)
- for each arrangement found we check if it is strictly increasing

Math model:

replaceNodeonOdd(l1...lm, LEVEL, E) = ∅ if m = 0
E ∪ replaceNodeonOdd(l2...lm, LEVEL, E), if l1 = E and l1 atom and LEVEL mod 2 = 1
l1 ∪ replaceNodeonOdd(l2...lm, LEVEL, E), if l1 atom and LEVEL mod 2 = 1
replaceNodeonOdd(l1, LEVEL, E) ∪ ... ∪ replaceNodeonOdd(lm, LEVEL, E), otherwise

```
(defun replaceNodeonOdd (L LEVEL E)
    (cond
        ((and (atom L) (eq (mod LEVEL 2) 1)) (list E))
        ((atom L) (list L))
        (t (list (mapcan #'(lambda (5) (replaceNodeonOdd 5
                                (+ 1 LEVEL) X)) L)))
    )
)

(replaceNodeonOdd '(a (b (g)) (c (d (e)) (f))) -1 'h))
```

we call the function with -1 because when it will first enter, L will be a list and therefore go on the last case, therefore increasing the level such that when mapcan is applied and "a" comes to the function, the level will be 0 (as required).

if we find ~~the~~ on atom ~~that we want~~ on the odd ~~level~~, we return the replace value, if we find an atom ~~that~~ is not on the level that we want (= not odd) and otherwise we apply mapcan for the entire list which will apply our function (a lambda function) for every element in the list and creates a list with the results returned from each node.

CÂMPEAN CĂTĂLIN ALEXANDRU