```
> #========== Lab 7 problems ==============================
> #Q3: Write a procedure that takes a list with positive integers between 1 and 100
  # and returns a list of size 10 containing the counts in the ranges 1 –10,11 –2 0,21 –30,..,
      91 –100.
  #For example, when called on the list [11, 21, 31, 41, 51, 53] it should return
  #` `[0, 1, 1, 1, 1, 2, 0, 0, 0, 0]. Feel free #to use the ceil function in this question.

> myHist := proc(L :: list)
   local num, Lcount;
   Lcount := [seq(0, i = 1 ..10)];
   for num from 1 to nops(L) do
   Lcount[ceil(L[num]/10)] := Lcount[ceil(L[num]/10)] + 1 ;
   end do;
   return Lcount;
   end proc;
```

```
myHist := proc(L::list)                                                    (1)
    local num, Lcount;
    Lcount := [seq(0, i = 1 ..10)];
    for num to nops(L) do
        Lcount[ceil(1/10*L[num])] := Lcount[ceil(1/10*L[num])] + 1
    end do;
    return Lcount
end proc
```

```
> #==================================
  # Write a procedure that takes a sorted list of integers as input
  #  and returns the number of distinct elements in the list.  You are
  #NOT allowed to convert the list to a set for this question.  Test your
  # procedure with lists in ascending  order as well as lists in descending order.
  #==================================
  numDistinct := proc(L :: list)
   local num, count;
   count := 1;
   for num from 2 to nops(L) do
   if L[num] ≠ L[num − 1] then
   count := count + 1 ;
   end if;
   end do;
   return count;
   end proc;
```

```
numDistinct := proc(L::list)                                               (2)
    local num, count;
    count := 1;
    for num from 2 to nops(L) do
        if L[num] <> L[num − 1] then count := count + 1 end if
    end do;
    return count
```

```
end proc
```

```
> numDistinct([1, 1, 2, 3, 4, 4, 5, 5, 5]);
```

$$5 \tag{3}$$

```
> #======= reversing a list ==================
  recrev := proc(L :: list, n :: posint)#reverse list L[1..n]
   local LL;
   if n = 1 then return [L[1]];
   else
   #print(recrev(L, n - 1 ));
   LL := [L[n], op(recrev(L, n - 1))];
   return LL;
   end if;

   end proc;
```

$$recrev := \mathbf{proc}(L::list, n::posint) \quad \mathbf{local}\ LL; \quad \mathbf{if}\ n = 1\ \mathbf{then return}\ [L[1]]\ \mathbf{else}\ LL := [L[n], op(recrev(L, n-1))];\ \mathbf{return}\ LL\ \mathbf{end if} \quad \mathbf{end proc} \tag{4}$$

```
>
>
>
> trace(recrev);
```

$$recrev \tag{5}$$

```
> recrev([1, 2, 3, 4, 5], 5);
{--> enter recrev, args = [1, 2, 3, 4, 5], 5
{--> enter recrev, args = [1, 2, 3, 4, 5], 4
{--> enter recrev, args = [1, 2, 3, 4, 5], 3
{--> enter recrev, args = [1, 2, 3, 4, 5], 2
{--> enter recrev, args = [1, 2, 3, 4, 5], 1
<-- exit recrev (now in recrev) = [1]}
                              LL := [2, 1]
<-- exit recrev (now in recrev) = [2, 1]}
                              LL := [3, 2, 1]
<-- exit recrev (now in recrev) = [3, 2, 1]}
                              LL := [4, 3, 2, 1]
<-- exit recrev (now in recrev) = [4, 3, 2, 1]}
                              LL := [5, 4, 3, 2, 1]
<-- exit recrev (now at top level) = [5, 4, 3, 2, 1]}
                              [5, 4, 3, 2, 1]
```

(6)

```
> untrace(fib) :
>
>
```