ECS 140A: Summer Session 1 2021
Homework Assignment 3

Due Date: No later than Thursday, July 15, 11:00pm.

For each of the following problems, you are to provide two solutions: one using the pattern-matching techniques discussed in Episode 18, and one not using those pattern-matching techniques (i.e., the ones we discussed before talking about pattern-matching over lists). For example, the two solutions for myappend might look like:

```
-- without pattern matching
myappend list1 list2
  | list1 == [] = list2
  | otherwise   = (head list1):(myappend (tail list1) list2)

-- with pattern matching
myappend_pm [] list2     = list2
myappend_pm (x:xs) list2 = x:(myappend_pm xs list2)
```

Note that I have ended the name of the pattern-matching solution with "_pm". Please do the same for all the pattern-matching solutions that you submit for this assignment.

To implement the functions below, you may use only the following list operations -- ':', 'head', 'tail', 'null', and 'elem' . (Note that there are pattern matching equivalents for most of these.) Do not resort to just giving a new name to an existing Haskell function that already does what we want your function to do. So, for example

```
myappend inlist1 inlist2 = inlist1 ++ inlist2
```

wouldn't get you any points.

Please make sure you name your functions with the names provided below (with and without the _pm suffix). **Also, include type declarations with your functions. It will be good practice.**

Submit your solutions as a single file named "hw3.hs".

Grading will be on a 3-point scale for each solution (7 problems x 2 solutions per problem x 3 points maximum per solution = 42 points maximum).

And now, here are your homework problems:

1) myremoveduplicates

```
myremoveduplicates "abacad" => "bcad"
myremoveduplicates [3,2,1,3,2,2,1,1] => [3,2,1]
```

2) myintersection

For this function, the order of the elements in the list returned by the function doesn't matter. Also, if the arguments to the function have duplicate elements in the list, then the result of the intersection is unspecified.

```
myintersection "abc" "bcd" => "bc"
myintersection [3,4,2,1] [5,4,1,6,2] => [4,2,1]
myintersection [] [1,2,3] => []
myintersection "abc" "" => ""
```

3) mynthtail

```
mynthtail 0 "abcd" => "abcd"
mynthtail 1 "abcd" => "bcd"
mynthtail 2 "abcd" => "cd"
mynthtail 3 "abcd" => "d"
mynthtail 4 "abcd" => ""
mynthtail 2 [1, 2, 3, 4] => [3,4]
mynthtail 4 [1, 2, 3, 4] => []
```

4) mylast

```
mylast "" => ""
mylast "b" => "b"
mylast "abcd" => "d"
mylast [1, 2, 3, 4] => [4]
mylast [] => []
```

5) myreverse

There's a simple but inefficient solution to this problem, and a much more efficient solution. For full credit, provide the more efficient solution.

```
myreverse "" => ""
myreverse "abc" => "cba"
myreverse [1, 2, 3] => [3, 2, 1]
myreverse [] => []
```

6) myreplaceall

```
myreplaceall 3 7 [7,0,7,1,7,2,7] => [3,0,3,1,3,2,3]
myreplaceall 'x' 'a' "" => ""
myreplaceall 'x' 'a' "abacad" => "xbxcxd"
```

7) myordered

```
myordered [] => True
myordered [1] => True
myordered [1,2] => True
myordered [1,1] => True
myordered [2,1] => False
myordered "abcdefg" => True
myordered "ba" => False
```