Problem Set 7

My belt holds my pants up, but the belt loops hold my belt up.
Who is the real hero?
— *Mitch Hedberg*

- This problem set is due **at 9:00am** on **March 31, 2025** .
- This problem set comprises 2 problems.
- Submit early. Do **not** wait till the last moment.
- Each solution should start on a new page.
- We will give full credit **only** for correct solutions that are described clearly and convincingly.

Problem 7-1. PPT Question [60 points]

In this problem we will rewrite our favourite higher-order functions using for loops. Once you write them as loops, use loop invariants to formally prove that the loop implementation is equivalent to the given recursive implementation.

(a) [20 points] Implement the fold function using a for loop:

```
let rec fold (f : 'a -> 'b -> 'b)
    (v : 'b) (l : 'a list) : 'b =
  match l with
  | [] -> v
  | h :: t -> f h (fold f v t)
;;
```

(b) [20 points] Implement the unfold function using a for loop:

```
let rec unfold (n : int) (f : int -> 'a)
    (l : 'a list) : 'a list =
  match n with
  | 0 -> l
  | _ -> unfold (n - 1) f (f (n - 1) :: l)
;;
```

(c) [20 points] Implement the exists function using a for loop:

```
let rec exists (p : 'a -> bool) (l : 'a list) : bool =
  match l with
  | [] -> false
  | h :: t -> match (p h) with
              | true -> true
              | false -> exists p t
;;
```

Remember to prove the equivalence of the functions using loop invariants.

Problem 7-2. Sorting a List [40 points]

You are given an implementation of a sorting algorithm that utilizes helper functions such as `min_elem` and `remove`. This implementation follows a recursive approach to sorting an integer list. However, your task is to rewrite this sorting exclusively for loops (that is without any recursive functions or any recursive helper functions).

Once you have implemented the iterative version of the sorting algorithm, you must prove that for any given integer list, your implementation produces a sorted output. Specifically, you need to show that the output satisfies the following property:

For any list $l = [x_1, x_2, \cdots, x_n]$, the resulting sorted list $s = [y_1, y_2, \cdots, y_n]$ must satisfy these two properties:

1. The list s is a permutation of l (i.e., it contains exactly the same elements, with the same multiplicities).
2. The list s is in non-decreasing order, meaning that for all indices i such that $1 \leq i < n$, we have $y_i \leq y_{i+1}$.

Your proof should clearly establish that the transformation from l to s maintains these properties for valid inputs. Here is the code for sorting:

```
let rec min_elem (l : int list) : int =
  match l with
  | [] -> failwith "Empty list has no minimum"
  | [x] -> x
  | h :: t ->
      let min_t = min_elem t in
      if h <= min_t then h else min_t
;;

let rec remove (a : int) (l : int list) : int list =
  match l with
  | [] -> failwith "This list does not have the element"
  | h :: t -> if h = a then t else h :: (remove a t)
;;

let rec sort_min (l : int list) : int list =
  match l with
  | [] -> []
  | _  ->
      (min_elem l) :: sort_min (remove (min_elem l) l)
;;
```