COMP 3958: Lab 1

Put your implementation in a file named `lab1.ml`. Be sure to test your functions in `utop`. Provide comments to indicate what each function does. Your file must compile. If it does not, you may receive no credit for this lab exercise. Maximum score: 14.

Implement each of the following functions using recursion without calling any external function except those in `Stdlib` and the `List.rev` function.

For `zip`, `unzip` and `dedup`, provide both tail-recursive and non-tail-recursive implementations. The name of the tail-recursive version should end in `_tr`, e.g., the two "zip" functions should be named `zip` and zip_tr. You may implement additional helper functions if necessary.

Provide at least 3 tests for each implementation.

1. `val drop :  int -> 'a list -> 'a list`

   `drop n l` returns `l` with the first `n` elements dropped. For example,

   ```
   drop 3 [4;2;6;7;6;8;1] returns [7;6;8;1]
   drop (-1) [3; 2; 7] returns [3;2;7]
   drop 4 [3;2;7] returns []
   ```

2. `val zip :  'a list -> 'b list -> ('a * 'b) list`

   `zip lst1 lst2` returns a list consisting of pairs of corresponding elements of `lst1` and `lst2`. If `lst1` and `lst2` are of different lengths, the function stops "zipping" when the shorter list ends. For example,

   ```
   zip [1; 2; 3] ['a'; 'b'; 'c'; 'd'] returns [(1,'a'); (2,'b'); (3,'c')].
   ```

3. `val unzip :  ('a * 'b) list -> 'a list * 'b list`

   `unzip` takes a list `lst` of pairs and returns a pair of lists where the first list consists of the first element of each pair in `lst` and the second list consists of the second element of each pair in `lst`. For example,

   ```
   unzip [(1,'a'); (2,'b'); (3,'c')] returns ([1; 2; 3], ['a'; 'b'; 'c'])
   ```

4. `val dedup:  'a list -> 'a list`

   `dedup lst` returns a list where all consecutive duplicated elements in `lst` are collapsed into a single element. For example,

   ```
   dedup [1; 1; 2; 3; 3; 3; 2; 1; 1] returns [1; 2; 3; 2; 1]
   ```