CSC301 Assignment #6
Baheem

To receive credit for these exercises you have to use **pattern matching** in your function definitions. Submit your source code and proof that your code works via Brightspace.

For each of these exercises from Chapter 7, include your code in a copy/paste-able form and a screen capture (.docx, .pdf, .gif, .jpg, or .png) demonstrating the correct operation of your function in at least two situations. For exercise 2, you should use two different types and in one instance return true; the other, false. For exercise 6, variations could/ should include negative numbers, duplicates, already-sorted data, etc. While both functions should support empty list arguments, do not count that as one of your minimum number of examples.

Exercise
Write a function *more* of type *int * (int list) -> int list* so that *more(e,L)* returns a list of elements of *L* that are strictly larger than e.

```
1  fun more (e, L) =
2      if null L
3          then false
4          else if e = hd(L)
5                  then true
6                  else more (e, tl(L));
7  fun more (e, nil) = false
8  |    more(e, x::xs) = if (e=x)
9                          then true
10                         else more(e, xs);
11
12 more (1, []);
13 more (1, [1, 2]);
14 more (1, [2, 1, 3]);
15 more (1, [2, 3]);
```

```
val more = fn : ''a * ''a list -> bool
stdIn:8.28 Warning: calling polyEqual
val more = fn : ''a * ''a list -> bool
val it = false : bool
val it = true : bool
val it = true : bool
val it = false : bool
-
```

Exercise
Write a *quicksort* function of type *int list -> int list*. Quicksort works very similar to the mergesort but instead of halving lists the lists are split according to a pivot element. Here is a review of the quicksort algorithm: First pick an element on the input list and call it the pivot (the first element of the list is usually a good choice). Partition the rest of the list into two sublists: one with all the elements less than the pivot and one with all the elements not less than the pivot. Recursively sort the sublists. Combine the sorted sublists and the pivot into the final sorted list.

```
1  fun qsort([]) = []
2  |   qsort(f, pivot::rest) =
3      let fun split([], ys, zs) = qsort(ys) @ [pivot] @ qsort(zs)
4      |   split(x::xs, ys, zs) =
5                              if x <= pivot then split(xs, x::ys, zs)
6                              else split(xs, ys, x::zs)
7                  in split(rest, [], [])
8                   end;
```

```
- val qsort = fn : int list -> int list
-
```