CpSc 418 **Homework 1** Due: Jan. 18, 2016, 11:59pm

85 points.
5% extra credit if you submit your solutions by 11:59pm on January. 15.

Please submit your solution using the handin program. Submit the your solution as
cs418 hw1
Your submission should consist of the file `hw1.erl`. You may optionally include a file, `hw1-test.erl`, or you may include your test cases in `hw1.erl`. You should also include a file called `hw1.pdf` or `hw1.txt` to answer the non-coding questions: 1d, 2b, and 2d.

# The Questions

1. **Searching a list** (35 points).

   (a) `nthtail` (10 points)
   Write an Erlang function `nthtail(N, List) -> Tail` that returns the result of `N` applications of `tl` to `List`. For example:

   ```
   code(1, [1,2,3]) -> tl([1,2,3]) -> [2,3];
   code(2, [1,2,3]) -> tl(tl([1,2,3])) -> [3];
   code(3, [1,2,3]) -> tl(tl(tl([1,2,3]))) -> [];
   code(0, [1,2,3]) -> [1,2,3];
   ```

   `nthtail(N, List)` should fail, if `N` is not an integer; `N` is negative; `N` is greater than `length(List)`; or `List` is not a list. This is the same functionality as `lists:nthtail`. You **may not** use `lists:nthtail` to implement your `nthtail` function, but you are very welcome to use `lists:nthtail` for test cases.

   (b) `prefix` (10 points)
   Write an Erlang function `prefix(List1, List2) -> boolean()` that returns true iff `List1` is a prefix of `List2`. The fuction `prefix(X, Y)` should fail if either `X` or `Y` is not a list. For example:

   ```
   prefix([], [1,2,3]) -> true;
   prefix([1], [1,2,3]) -> true;
   prefix([1,2,3], [1,2,3]) -> true;
   prefix([1,2,3,4], [1,2,3]) -> false;
   prefix([3,2,1], [1,2,3]) -> false;
   prefix(banana, [1,2,3]).
   ** exception error: no function clause matching prefix(banana, [1,2,3])
   prefix("banana", "bananas") -> true
   prefix("alumna", "alumni") -> false
   ```

   This is the same functionality as `lists:prefix`. You **may not** use `lists:prefix` to implement your `prefix` function, but you are very welcome to use `lists:prefix` for test cases.

   (c) `search` (15 points)
   Write an Erlang function `search(List1, List2) -> [integer()]` that returns a list of all integers, `N`, such that `List1` is a prefix of `List2` starting at position `N`. For example:

   ```
   search([2,3], [1,2,3,4,3,2,1,2,3,4]) -> [2,8]
   search("an", "banana") -> [2,4]
   search("a", "banana") -> [2,4,6]
   search([], "banana") -> [1,2,3,4,5,6,7]
   ```

(d) **Just for fun** (0 points)
There are simple implementations of `search` that require $O($`length(List1)` $*$ `length(List2)`$)$ time in the worst-case. I'm expecting solutions like that for question 1c, even though they are asymptotically inefficient. Can you do better? Hint: consider the case where `Length(List1)` $\ll$ `Length(List2)`. Implement your solution in Erlang and run experiments to compare the runtime of your improved algorithm with the simple one.

2. **List subtraction.** (50 points)
The Erlang operator `--` computes the "difference" between two lists. It is implemented by the function `lists:subtract`. As noted in the documentation, the `--` operator can be quite slow if the lists it is operating on are long.

   (a) (10 points) Let `L1 = lists:seq(1, N)` and `L2 = lists:reverse(L1)`. Measure the run time for `L1 -- L2` for `N=1000`, `N=2000`, `N=3000`, `N=5000`, `N=10000`, `N=20000`, `N=30000`, and `N=50000`. Try other values for `N` if you think that will be helpful to understand the behaviour of `--`.

   (b) (10 points) Does the run-time appear to be linear, $N \log N$, quadratic, exponential, or something else (if so, what)? Justify your answer based on your data reported for question2a.

   (c) (20 points) Write `subtract(List1, List2) -> List3` "fast" version of list difference. You `subtract` function should behave like `lists:subtract`, just faster, at least if the argument lists are long. Your implementation should run in time $O(N \log N)$ time where $N = \max($`length(L1)`$,$ `length(L2)`$)$. You may use any functions you like from the Erlang library. In particular `lists:seq`, `lists:sort`, `lists:unzip`, and `lists:zip` may be helpful.

   (d) (10 points) Experimentally verify the $O(N \log N)$ run time.

# Why?

**Question 1**: This question is to give you some more experience writing Erlang functions, including using recursion, pattern matching and guards. The first two parts, `nthtail` and `prefix` are supposed to be easy – **don't make them complicated**. These are especially for students who have little or no prior background with functional languages – these functions should be easy to write and test. For everyone, these are also an exercise in good coding – make sure you include the guards so these fail right away on bad inputs. Recursing forever or failing after extensive computation are symptoms of poor software design.

The `search` function is intended to make you think a bit more about how to implement the requested behaviour. Once you've thought it through, your code should be simple. I wrote a one-line, list comprehension for the body of `search`. If you find comprehensions to be incomprehensible, I also wrote a recursive version (with a helper function). It's about ten lines of code. The annoying part was a couple of cases to avoid calling `tl` when `List2` is empty.

The "just for fun" part is for those who have enough functional programming experience that the first two parts were boring. You can think about this, and have fun writing a better algorithm. If you've had an advanced algorithms course, you might find this question obvious as well – *c'est la vie*.

**Question 2**: This is an opportunity to do something a little more involved using functions, lists, tuples, and pattern matching. It also provides some practice with measuring run times and interpreting the results. This question is intended to be more challenging than question 1, but it doesn't require huge amounts of code. My solution has two functions: `subtract` and a helper. My implementation of `subtract` is five lines of Erlang, but you could easily use a few more or a few less. If you wanted, you could write the whole body of the function as a single expression – "Look, mom. No commas!".

My helper function consists of four, one-line pattern matching cases. It's basically a variation on the ideas from `ordsets:subtract/2`.

My notes about my implementations are just to keep you from getting stuck writing overly complicated solutions. Of course, some solutions will be shorter than mine, some longer, some more efficient or more elegant. That's OK. If your solution is turning into something way longer or way more complicated than what I described, ask for help.