CpSc 418 **Homework 1** Solution Set

1. `closest(P, PointList)`.
   My Answer: see http://www.ugrad.cs.ubc.ca:~cs418/2016-2/hw/1/sol/hw1sol.erl.

2. `allTails(L)`, part 1 **(10 points)**.

   (a) **(8 points)** Write a tail-recursive implementation of `allTails`.
   My Answer: see hw1sol.erl.

   (b) **(2 points)** Print the value produced by `allTails(lists:seq(1000, 1010))`.
   My Answer:

```
[[],
  [1010],
  [1009,1010],
  [1008,1009,1010],
  [1007,1008,1009,1010],
  [1006,1007,1008,1009,1010],
  [1005,1006,1007,1008,1009,1010],
  [1004,1005,1006,1007,1008,1009,1010],
  [1003,1004,1005,1006,1007,1008,1009,1010],
  [1002,1003,1004,1005,1006,1007,1008,1009,1010],
  [1001,1002,1003,1004,1005,1006,1007,1008,1009,1010],
  [1000,1001,1002,1003,1004,1005,1006,1007,1008,1009,1010]]
```

3. `allTails(L)`, part 2 **(10 points)**.
   Evaluate the Erlang expression below:

```
[     {N, erts_debug:size(hw1:allTails(lists:seq(1,N)))}
   || N <- lists:seq(100,1000,100)
]
```

   My Answer: I got

```
[100,402,  200,802,  300,1202, 400,1602, 500,2002,
 600,2402, 700,2802, 800,3202, 900,3602, 1000,4002]
```

   (a) **(5 points)** Does the memory used for `allTails` grow linearly, quadratically, exponentially, or as some other function of `N`? Explain why this is the case, in terms of how Erlang apparently implements the list operations in your `allTails` function. You should explain the "apparently" part – what did you infer from the data?

   My Answer: Linearly. The number of words used to store `alltails(lists:seq(1,N))` is exactly $4N + 2$ for the eleven values of `N` above.
   This suggests that Erlang represents lists using linked lists. Each list cell appears to be represented with two "words" – a pointer to the value for the current element, and a pointer to the next list cell. Thus, `lists:seq(1,N)` should produce a list that is stored using $2N$ words. A simple experiment confirms this. I ran the same command as suggest above without the `hw1:allTails` part. This must reports the memory usage for the lists returned by lists:seq:

```
[     {N, erts_debug:size(lists:seq(1,N))}
   || N <- lists:seq(100,1000,100)
]
```

   I got:

```
[100,200,  200,400,  300,600,  400,800,  500,1000,
 600,1200, 700,1400, 800,1600, 900,1800, 1000,2000]
```

Each time `allTails` matches `[Head | Tail]` to `List`, `Head` appears to be bound to a pointer to the value field of the first list cell of `List`, and `Tail` appears to be bound to a pointer to the next list cell in `List`. Thus, each list cell of `allTails` consists of two pointers and is stored using two Erlang words. This is 2`N` words in addition to the 2`N` words for the list returned by lists:seq. This accounts for the 4`N` words part of the linear term.

Where does the +2 part of the linear formula come from. I haven't tracked that down. Simplicity seems to be a major objective of the Erlang memory system. I'll guess that it uses reference counting to find the "obvious" garbage. In fact, I can't think of a way to create a cycle of references in Erlang; so reference counting may be sufficient. Constructing `allTails` created two ways to get to the list from lists:seq. This may require a couple of words for reference counting, but I haven't tried anything to test that guess.

Note: my answer is way more detailed than you need for full credit. You just need to conclude that the list created by `allTails` creates pointers (or references) to the original list and doesn't make copies. **This is a big deal.** Why? Because it means that when your code can use a list or a suffix of a list, making the reference is really fast. Operations that modify the end of a list, for example,

```
LongList ++ [OneMoreElement]
```

tend to be very inefficient because they require copying `LongList` to create a separate version where the final `next`-pointer can be modified. Avoid appending to lists.

(b) Draw a plot, or semi-log plot, or log-log plot of your data to support your answer from part (a). Include a line of best fit.

My Answer: As noted in my answer for part (a), the size of `allTails(lists:seq(1,N))` is *exactly*

$$\texttt{size} \quad = \quad 4N + 2$$

That's my line of best fit. Figure **??** is my plot of the data an my line. The matlab code for this plot is at
`http://www.ugrad.cs.ubc.ca/~cs418/2016-2/hw/1/sol/hw1q3.m`
Some people asked me if they really had to do this if the fit is so obvious. In this case, yes. There will be other problems where you need to describe run-time trends, or other trends. I'll often get answers that take *anything* that grows faster than linear and call it "exponential". Don't do that. If you draw the plot, it becomes if you labeled the trend wrong. For this problem, I'd like to make it

> Draw the plot if you might confuse "quadratic" and "exponential" on a future homework assignment.

I don't have a way to predict what each person will do on future assignments; so, for this one, I asked everyone to draw a plot and include it in their solution.

4. `longestOverlap(L1, L2)`
   My Answer: see hw1sol.erl.

5. **Test cases (5 points):**
   My Answer: see `http://www.ugrad.cs.ubc.ca:~cs418/2016-2/hw/1/sol/hw1_test.erl`.

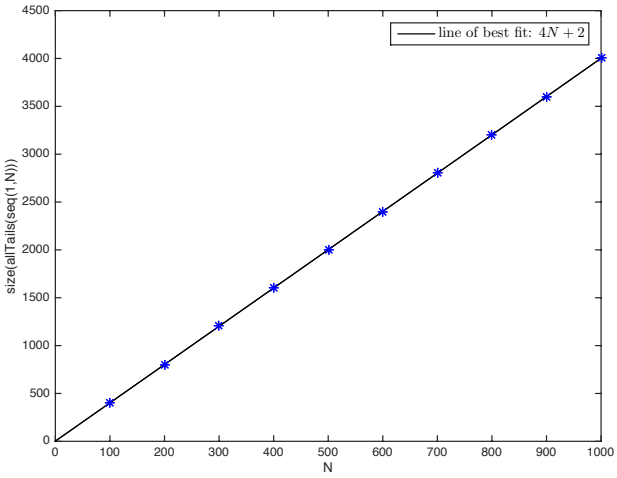

Figure 1: Size of `allTails` and line of best fit