CpSc 418 **Homework 2** Solution Set

1. **Getting there faster (15 points + 5 Extra Credit)**
   Solution in hw2.erl.

2. **Run-Length Encoding (10 points)**
   Solution in hw2.erl.

3. **Longest Run (20 points)**
   Solution in hw2.erl.

4. **Sequence Matching (40 points)**

   (a) **(5 points)** Write a sequential function,

   ```
   best_match(L1, L2) -> {MatchCount, Alignment}
   ```

   that computes `MatchCount` and `Alignment` as described above.

   Solution in hw2.erl.

   (b) **(10 points)** What is the run-time for your implementation of `best_match` in terms of $N_1 =$ `length(L1)` and $N_2 =$ `length(L2)`. Derive and justify a big-O formula. Measure the run-time of your implementation using the function `time_it:t` in the class Erlang library (i.e. get averages and standard deviation over at least 20 runs for each data point). Compare your empirical measurements with your analytical formula.

   Solution: My solution from part (a) works by computing the match for every possible alignment of `L1` and `L2`. There are `|L1| + |L2| - 1` such possible alignments, and each requires $O(|L1|)$ work to count the number of matching positions. This gives a sequential runtime that is $O(|L1|(|L1|+|L2|))$. This is an acceptable answer.
   With some more care, we can show a tighter bound of $O(|L1||L2|)$. First, I'll note that the runtime is proportional to the number of positions at which comparisons are performed. This is just the sum of the amount for each alignment. First, consider the case when $|L1| \leq |L2|$, Let k denote the alignment. we get:

   If $1 - |L1| < k < 0$, then $|L1|$ and $|L2|$ overlap in $|L1| + k$ positions. The total number of comparisons for all $|L1| - 1$ such alignments is

   $$\begin{array}{ll} \sum_{k=-1}^{1-|L1|} |L1| + k & \\ = & \sum_{j=1}^{|L1|-1} j \\ = & \frac{1}{2}|L1|(|L1| - 1) \end{array}$$

   If $0 \leq k \leq |L2| - |L1|$, then $|L1|$ and $|L2|$ overlap in $|L1|$ locations , and there are $|L2| - |L1| + 1$ such alignments. This results in a total of $|L1|(|L2| - |L1| + 1)$ comparisons.

   If $|L2|-|L1|k < |L2|$, there are $|L1|-1$ such alignments, and the total number of comparisons is $\frac{1}{2}|L1|(|L1|-1)$ by an argument

   Taking the sum of the number of comparisons for these three cases yields a total of $|L1||L2|$ comparisons for the complete computation.
   When $|L1| \geq |L2|$ a similar argument applies, and the algorithm performs $|L1||L2|$ comparisons.

   $\therefore$ the run time for the sequential algorithm is $O(|L1||L2|)$.

   Grading note: the solution in hw2.erl performs a `tl(L1)` with each recursive call of `bm_neg` and a `tl(L2)` with each recursive call of `bm_pos`. A slower way to get the same functionality would be to call `lists:nthtail(L1, -Align)` or `lists:nthtail(L2, Align)` in `bm_neg` or `bm_pos` or their equivalents in solution. This increases the run time to $O((|L1| + |L2|)^2)$. While such a solution satisfies, Ror any solution, the analysis should apply to the code submitted.

   Now, we're ready for the empirical measurements. I added a function `time_bm_seq` to `hw2_test` to measure the run-time for list `L1` with lengths for 10 to 100 and list `L2` with lengths from 10K to 100K. I ran all tests

on `thetis.cs.ubc.ca` because I've found that it gives some of the most consistent timing measurements of various machines tht I've tried. The function `time_it:t` can be determine how many times to call the function under test to meet a time bound (default 1.0 seconds) or a specified number of calls. I observed that the reported time was less than 0.02 seconds (thus 50 or more calls) if $|L1||L2| < 1,600,000$. To satisfy the requirement that each trial be an average of at least 50 runs, my code calls `time_it:t` with the default (run for one second), if $|L1||L2| < 1,600,000$, and specifies 50 runs if $|L1||L2|$ is greater than this bound. The table below shows the timing measurments, times are reported in seconds:

	$\|L2\|$									
$\|L1\|$	10K	20K	30K	40K	50K	60K	70K	80K	90K	100K
10	0.0016	0.0032	0.0047	0.0063	0.0079	0.0094	0.0110	0.0126	0.0141	0.0157
20	0.0027	0.0055	0.0081	0.0107	0.0134	0.0161	0.0187	0.0214	0.0242	0.0268
30	0.0038	0.0076	0.0114	0.0151	0.0190	0.0227	0.0264	0.0304	0.0347	0.0380
40	0.0050	0.0098	0.0147	0.0196	0.0245	0.0293	0.0346	0.0392	0.0439	0.0502
50	0.0060	0.0121	0.0182	0.0242	0.0299	0.0367	0.0420	0.0492	0.0539	0.0600
60	0.0074	0.0143	0.0214	0.0289	0.0355	0.0442	0.0501	0.0569	0.0650	0.0708
70	0.0085	0.0165	0.0246	0.0328	0.0410	0.0492	0.0573	0.0655	0.0748	0.0819
80	0.0097	0.0187	0.0284	0.0372	0.0465	0.0567	0.0654	0.0743	0.0835	0.0932
90	0.0106	0.0209	0.0313	0.0417	0.0522	0.0626	0.0738	0.0831	0.0936	0.1040
100	0.0116	0.0238	0.0346	0.0460	0.0577	0.0691	0.0815	0.0919	0.1030	0.1150

From our analysis, we expect these values to be roughly proportional to $|L1||L2|$, especially for larger values of $|L1|$ and $|L2$. To check this, I divide each entry by $|L1||L2|$ for that entry, and defined α to be the average of the quotients. I obtained a value of $\alpha = 12.5$ns. See best_match_seq.m for my Matlab code for the calculations reported here. I then divided each of the run-times reported above by $\alpha|L1||L2|$ – if the fit were perfect, this would produce an array where each entry is exactly 1. I got:

	$\|L2\|$									
$\|L1\|$	10K	20K	30K	40K	50K	60K	70K	80K	90K	100K
10	1.265	1.261	1.260	1.261	1.259	1.257	1.258	1.261	1.255	1.257
20	1.077	1.107	1.076	1.071	1.073	1.074	1.070	1.071	1.077	1.073
30	1.017	1.016	1.014	1.008	1.014	1.010	1.007	1.014	1.029	1.014
40	0.991	0.982	0.981	0.981	0.981	0.978	0.990	0.981	0.976	1.005
50	0.961	0.969	0.972	0.969	0.958	0.980	0.961	0.985	0.959	0.961
60	0.982	0.954	0.952	0.964	0.948	0.983	0.955	0.949	0.964	0.945
70	0.972	0.944	0.938	0.938	0.938	0.938	0.936	0.937	0.951	0.937
80	0.972	0.936	0.948	0.931	0.931	0.946	0.935	0.930	0.929	0.933
90	0.943	0.930	0.928	0.928	0.929	0.928	0.938	0.924	0.925	0.925
100	0.929	0.953	0.924	0.921	0.924	0.922	0.932	0.920	0.916	0.921

The values are all within $[0.921, 1.265]$, i.e. an "error" bounded by $+27\%$ and -8%. Not bad. It's reasonable to guess that the reason that we don't get an exact fit is that we're doing asymptotic analysis which ignores lots of details to let us focus on the big picture. In particular, we're ignoring lower-order terms, such as functions where the number of calls is $|L1|$ or $|L2|$. Just for fun, I did a least-squares best fit for the model:

$$t(n_1, n_2) \quad = \quad \alpha_0 + \alpha_1 n_1 + \alpha_2 n_2 + \alpha_{12} n_1 n_2$$

where $n_1 = |L1|$ and $n_2 = |L2$. This produced $\alpha_0 = -148.7\mu$s, $\alpha_1 = 4.9\mu$s, $\alpha_2 = 51.22 \times 10^{-8}$ns, and $\alpha_{12} = 10.98$ns. I then divided the measured run-times by $t(n_1, n_2)$ according to these value for the αs. The results are:

	$\|L2\|$									
$\|L1\|$	10K	20K	30K	40K	50K	60K	70K	80K	90K	100K
10	1.046	1.010	0.998	0.994	0.989	0.985	0.985	0.986	0.980	0.981
20	1.012	1.031	0.998	0.993	0.994	0.994	0.989	0.990	0.995	0.992
30	1.002	1.000	0.999	0.992	0.999	0.994	0.991	0.999	1.013	0.999
40	1.000	0.996	0.996	0.997	0.997	0.994	1.007	0.998	0.994	1.023
50	0.984	1.000	1.006	1.004	0.993	1.017	0.998	1.023	0.996	0.998
60	1.016	0.997	0.998	1.013	0.996	1.034	1.005	0.999	1.015	0.995
70	1.013	0.995	0.993	0.995	0.996	0.997	0.995	0.996	1.011	0.997
80	1.018	0.993	1.010	0.994	0.995	1.012	1.002	0.996	0.995	1.000
90	0.992	0.992	0.995	0.996	0.999	0.999	1.011	0.996	0.998	0.998
100	0.980	1.021	0.994	0.994	0.999	0.997	1.009	0.996	0.993	0.998

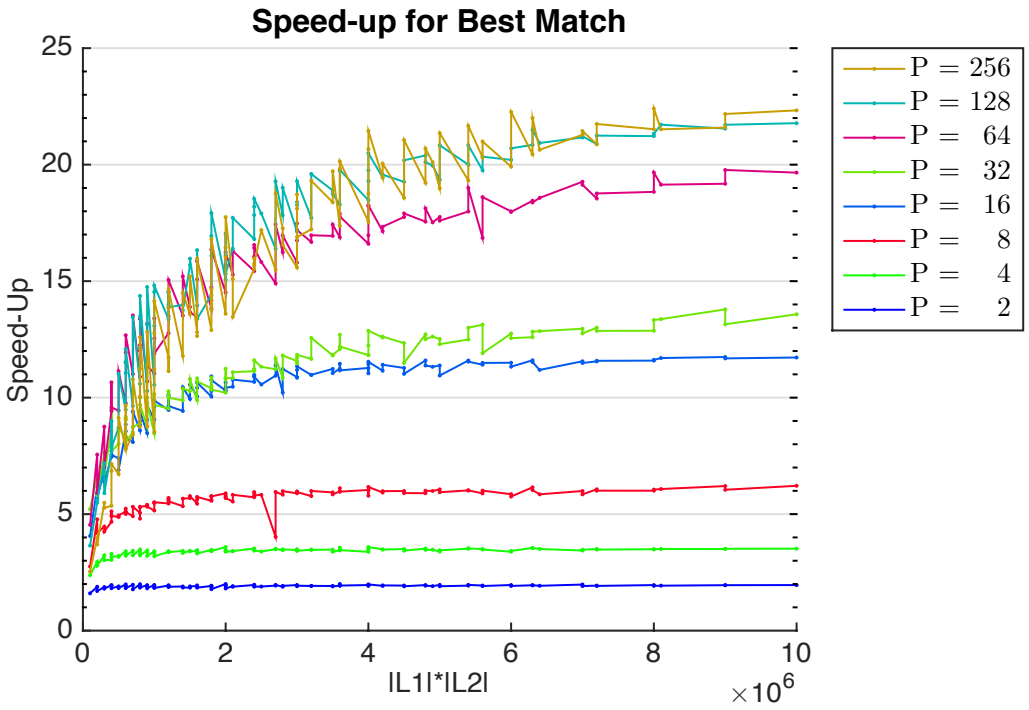

Figure 1: Speed-Up for `bm_par` as a function of $|L1| \cdot |L2|$ and P

In this case, the error is bounded between $+4.7\%$ and -2.1%, and for 74% of the test cases, the error is less than 1%. I'm happy with that.

(c) **Parallel Best Match (15 points)**
Solution in hw2.erl.

(d) **Measure the speed-up (10 points)**
I added functions `time_bm_par` and `time_bm_seq` to `hw2_test`. They are similar to the functions for measuring timings for the sequential version as described above. I ran them on `thetis` which has 32 cores, and each core is two-way multi-threaded. I ran the same sizes of `L1` and `L2` as for the sequential case, with trials with P (the number of processors) set to each power of 2 from 2 through 256. Figure 1 shows the speed-ups as a function of $|L1| \cdot |L2|$.

The minimum speed-up for these cases was 1.6; so, I ran some more trials with smaller values for $|L1|$ and $|L2|$. I plotted speed-up as a function of $|L1| \cdot |L2|$ with 2, 4, and 8 processors, $|L1| \in \{10, 20, 30, 50, 100\}$, and $|L2|$ from 100 to 1000 in steps of 100. Figures 2 and 3 show the results. In each case, it seems that the speed-up crosses 1, and thus the parallel version is faster than the sequential one, when $|L1| \cdot |L2|$ is between 5000 and 8000. I was surprised to see that the two processor case required the largest problem size to break even. I don't have a good explanation for that. I'll just note that the parallel version is faster when the product of the list lengths is greater than 8000.

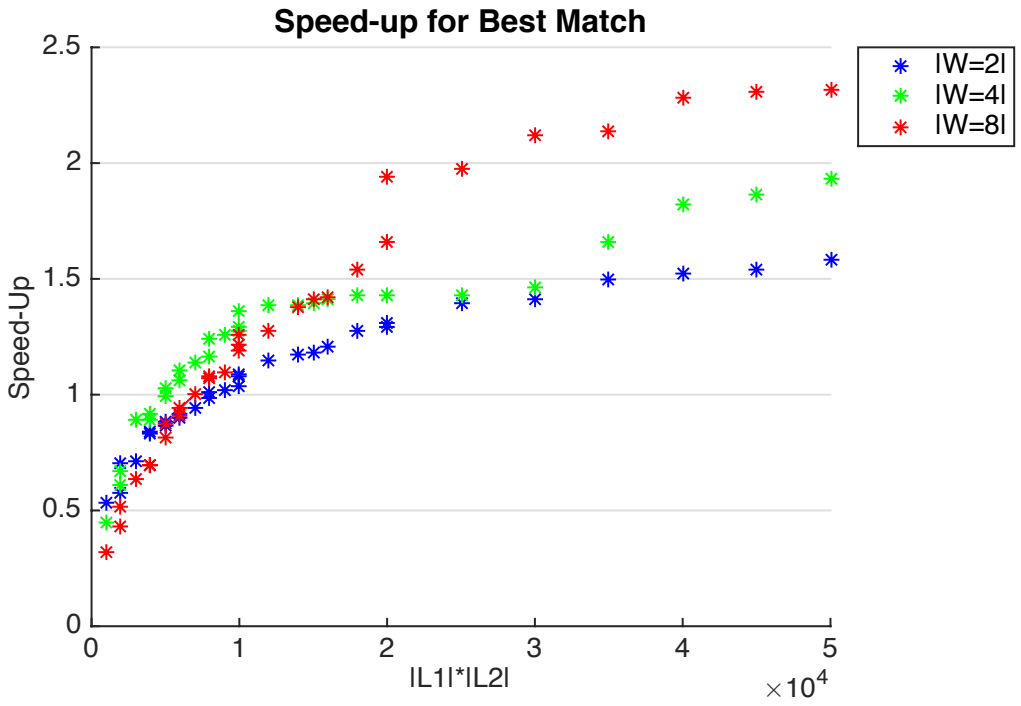

Figure 2: Speed-Up for `bm_par` as a function of $|L1| \cdot |L2|$ and P

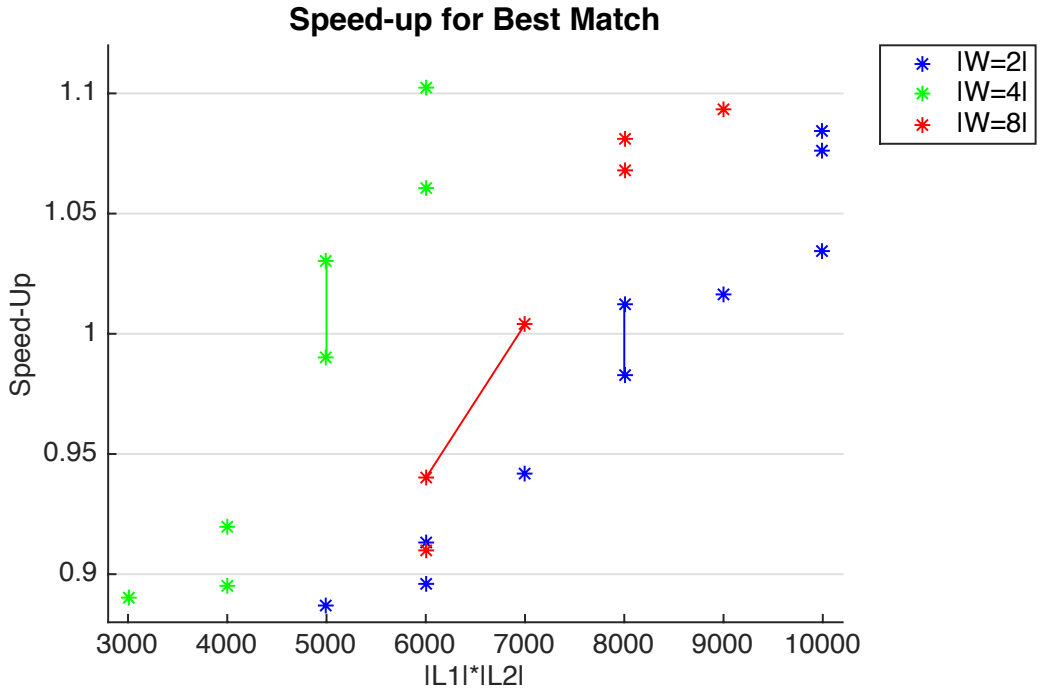

Figure 3: Speed-Up for `bm_par` as a function of $|L1| \cdot |L2|$ and P