CpSc 418 **Homework 4** Due: Nov. 14, 2018, 11:59pm
Early Bird: Nov. 13, 2018, 11:59pm

63 points.

Please submit your solution using the handin program. Submit your solution as
cs418 hw4
Your submission should consist of two files:

- `hw4.erl`: Erlang source code for the coding parts your solution.
- `hw4.pdf` PDF for the written response parts of your solution and the plots.

A templates for `hw4.erl` is available at
http://www.ugrad.cs.ubc.ca/~cs418/2018-1/hw/4/src/hw4.erl.

Please submit code that compiles without errors or warnings. If your code does not compile, we might give you zero points on all of the programming problems. If we fix your code to make it compile, we will take off lots of points for that service. If your code generates compiler warnings, we will take off points for that as well, but not as many as for code that doesn't compile successfully.

We will take off points for code that prints results unless we specifically asked for print-out. For this assignment, the functions you write should return the specified values, but they should not print anything to `stdout`. Using io:format when debugging is great, but you need to delete or comment-out such calls before submitting your solution. Printing an error message to `stdout` when your function is called with invalid arguments is acceptable but not required. Your code must fail with some kind of error when called with invalid arguments.

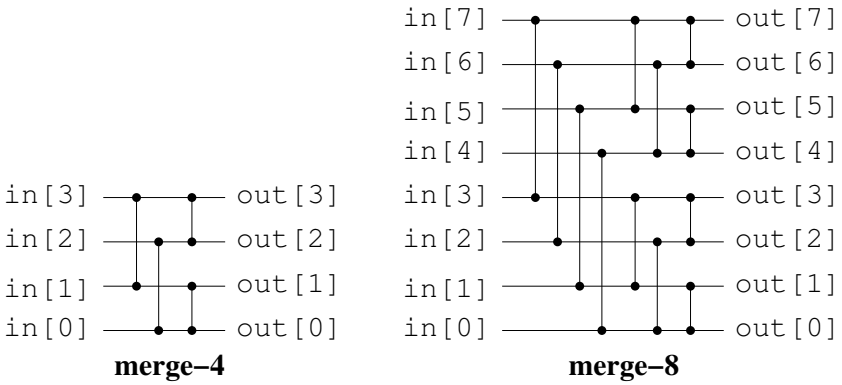

Figure 1: Bitonic Merge

1. Bitonic Merge **(28 points)**
   Figure 1 shows sorting networks for a 4-way and an 8-way bitonic merge. The TAs and I have been asked "What if the input to a merge network is *not* bitonic?". Good question. Let's find out.

   (a) Messing up a 4-way merge **(5 points)**
   Show that if the input to a 4-way, bitonic sorting network is not bitonic, then the output is not sorted. Give a short proof in your `hw4.pdf`.
   Hint: How many non-bitonic sequences of four 0s and 1s are there?

(b) Recognizing sorted lists **(5 points)**
Write a function, `is_sorted(List)`, that returns true if `List` is a list whose elements are in non-decreasing order. Note that `is_sorted([])` and
`is_sorted(SingletonList) when length(SingletonList) == 1` should both return `true`.

(c) Recognizing bitonic lists **(5 points)**
Write a function, `is_bitonic(List)`, that returns true if `List` is a bitonic list (up-down or down-up). Note that any list of three or fewer elements is bitonic.

(d) Lucking out with an 8-way merge **(5 points)**
Show that there is at least one non-bitonic inputs to an 8-way bitonic merge that produces a sorted output. You may do this with a written argument in your `hw4.pdf`, **or** you can do it by writing code in your `hw4.erl`. If you do the coding approach, write a short explanation of what you did in your `hw4.pdf`. Note: you'll need to do write code for the next part of this question no matter what you do here.

(e) More on the 8-way merge **(8 points)**
How many non-bitonic sequences of eight 0s and 1s are there? How many of them when input to an 8-way bitonic network produce a sorted output? Solve this problem by writing code in your `hw4.erl`. Write a brief explanation of how your code works. Hint: it should try all sequences of eight 0s and 1s and check the result of `bitonic_merge` on those that are not bitonic. The functions `hw4:find_combination/3` and `hw4:map_combination/3` should make it easier to write your solution.

2. Bitonic Sort **(35 points)**
`hw4:bitonic_sort(List)` is an implementation of bitonic sort for lists whose lengths are powers of two. Try it on a random list of `N` elements where `N` is a power of 2 (e.g. use misc:rlist) – it really works!

The function
`hw4:bitonic_sort(List, N, Dir)`
Divides `List` into segments of length `N` and sorts each in direction `Dir`. If `Dir > 0`, then the list is sorted into ascending order. If `Dir < 0`, then the list is sorted into descending order.

We want to find out if any of the compare-and-swap operations are unnecessary. The idea is to count how many compare-and-swap operations the algorithm has performed as it goes along. We'll introduce a variable, `CAS_count` for counting compare and swap operations. Then, we'll add a parameter `Target` so that the `Target`th compare-and-swap is disabled – i.e. it just copies its inputs to its outputs and never swaps them. Then, we can test the sorting network with each compare-and-swap disabled (one at a time) to find out if any are unneeded.

(a) Counting compare-and-swap operations. **(10 points)**
How many compare-and-swap operations are performed when executing `hw4:bitonic_sort(List)`? OK, we know the answer is supposed to be

$$\frac{N}{2}\left(\begin{array}{c}\log_2 N+1\\ 2\end{array}\right)$$

but let's write some code to make sure.

I've provided `bitonic_sort(List, N, Dir, CAS_count, Target)`. The parameter `CAS_count` is the total number of compare-and-swaps performed so far. The parameter `Target` gets passed to `bitonic_merge/5` which you will write, and you can pass it to `bitonic_step/5` if you like, but we won't use it any further until the next part of this question.

Your task for this question is to write `bitonic_merge/5` and `bitonic_step/5` to keep track of the total number of compare-and-swap operations that have been performed so far. When you are done,

```
bitonic_sort(List, length(List), +1, 0, 0)
```
should return a tuple of the form `{SortedList, CAS_count}` where `SortedList` is the sorted version of `List`, and `CAS_count` is the total number of compare and swap operations that were performed.

Nov. 11, 2018: To make it easier to run simple tests, I've added a function `bsort(List)` to the `hw4.erl` template that just calls
```
bitonic_sort(List, length(List), +1, 0, 0).
```

(b) Using the target. **(5 points)**
The functions `bitonic_sort/5`, `bitonic_merge/5`, and `bitonic_step/5` all have a parameter called `Target`. Modify `bitonic_step/5` so that the compare-and-swap is skipped (i.e. the input is copied directly to the output and never swapped) when performing the compare-and-swap operation that sets the total number done so far (i.e. `CAS_count`) to `Target`.

(c) Finding a bad input. **(10 points)**.
Complete the function, `find_bad_input(N, Target)`. If `bitonic_sort` sorts some input incorrectly when the `Target`th compare-and-swap is skipped, this function returns such an input. Otherwise, it returns `none`. Hint: the function `find_combination/3` is useful when solving this problem.

(d) Are all compare-and-swap operations needed? **(10 points)**
Complete the function, `useless_cas` that returns a list of all unnecesasry compare-and-swap operations. Are there any unnecessary compare-and-swap operations for a bitonic sorting network with 16 inputs?
Note: my solution takes about 20 seconds to run.