Homework 1

15-210: Parallel and Sequential Data Structures and Algorithms (Spring'12)

***due:** wed, jan 25 @ 11:59pm*

1 Introduction

This assignment is meant to help you familiarize yourself with the hand-in mechanism for 15-210 and get you thinking about proofs and coding again. To that end, you will answer some simple questions about the mechanics and infrastructure of the course, then implement three solutions to the stock market problem and perform some analysis of your solutions. Finally, you will do some exercises involving Big-O notation and prove some basic identities.

1.1 Submission

This assignment is distributed in a number of files in our `git` repository. Instructions on how to access that repository can be found at `http://www.cs.cmu.edu/~15210/resources/git.pdf`. This is how assignments will be distributed in this course.

This assignment requires that you submit both code and written answers.

All of your code for this assignment must be in the file `stock.sml`. Submit your solutions by placing these files in your handin directory, located at

```
/afs/andrew.cmu.edu/course/15/210/handin/<yourandrewid>/assn1/
```

Do not submit any other files. Name the files exactly as above. You can run the check script located at

```
/afs/andrew.cmu.edu/course/15/210/bin/check/01.sh
```

to make sure that you've handed in appropriate files.

Your written answers must be in a file called `hw01.pdf` and must be typeset. You do not have to use LaTeX, but if you do, we have provided the file `defs.tex` with some macros you may find helpful.

Your `stocks.sml` must contain all the code that you want to have graded for this assignment and must compile cleanly. If you have a function that happens to be named the same as one of the required functions but does not have the required type, it will not be graded.

1.2 Naming Modules

The questions below ask you to organized your solutions in a number of modules written from scratch. Your modules must be named exactly as stated in the handout. Correct code inside an incorrectly named structure, or a structure that does not ascribe to the specified signatures, *will not receive any credit*.

You may not modify any of the signatures or other library code that we give you. We will test your code against the signatures and libraries that we hand out, so if you modify the signatures your code will not compile and you will not receive credit.

1.3 The SML/NJ Build System

This assignment includes a substantial amount of library code spread over several files. The compilation of this is orchestrated by CM through the file `sources.cm`. Instructions on how to use CM can be found at on the website at `http://www.cs.cmu.edu/~15210/resources/cm.pdf`.

2 Mechanics

The following questions are intended to make sure that you have read and understood the various policies for the course, as well as found the tools that we've set up to communicate with you.

Task 2.1 (1%). What magic number is posted in the welcome message on the announce bboard?

Task 2.2 (3%). In each of the following situations, have the students followed or broken the collaboration policy? Briefly justify your answers with a sentence or two.

1. Ludmilla, Joaquin, and Alexander have lunch together after 210 lecture. Over lunch, they discuss the homework assignment released earlier in the week, including their progress so far. After lunch, all three go to different classes and don't think about the 210 homework until that evening.
2. While working on 213 homework near his friends from 210, Jon has a moment of insight into the 210 homework assignment. He becomes excited, and tells his friends what he thought of. Ishmael hasn't gotten that far in the homework, so he doesn't quite understand what Jon is talking about. Nonetheless, Ishmael copies down what he thinks are the key bits of what he heard to look at when he gets that far.
3. Yelena has been working on the 210 homework but can't figure out why her solution isn't compiling. She asks Abida to work through a couple of toy examples of functor syntax with together, and they do so with a text editor and the SML REPL. Afterwards, Yelena gets her homework to compile and keeps working towards a solution.

Task 2.3 (1%). If you hand in your homework 25 hours after the posted due date, and you have no late days remaining, what is your maxiumum possible score?

3 The Stock Market Problem

The problem with the stock market is that, while it is possible to make a great deal of money buying and selling stocks, it's easy to lose even more. The long-standing—if somewhat unhelpful—maxim to make more money than you lose is "buy low, sell high."

The stock market problem is finding the best opportunity to follow this advice: for any sequence of integer prices, where the index in the sequence represents time, find maximum jump from an earlier price to a later price. For example, if the sequence of prices was

$$\langle 40, 20, 0, 0, 0, 1, 3, 3, 0, 0, 9, 21 \rangle$$

then the maximum jump is 21, which happens between the price at time 2 and time 11. More formally, if `s` : `int seq`, the stock market problem is to compute the *maximum jump*

$$\max\{s_j - s_i \mid 0 \le i \le j < |s|\}.$$

The return type of the function you will implement is of type `stockres`, defined and passed to your functor in a structure that ascribes to `STOCK_PACKAGE`. The type `stockres` is given by

```
datatype stockres = INVALID | NOBUY | JUMP of int
```

Your implementations should evaluate to `INVALID` on an empty input, `NOBUY` if the input is decreasing or static, and `JUMP(x)` on input where `x` is the value of the maximum jump.

You will implement three solutions to this problem in the file `stocks.sml`. Each solution will be a functor ascribing to the signature `STOCKS`, defined in `STOCKS.sml`. Each functor will take a structure ascribing to `STOCK_PACKAGE` as its only argument. The signature `STOCKS` is defined in terms of the `SEQUENCE` signature from our library, which is documented in `http://www.cs.cmu.edu/~15210/resources/docs.pdf`. For testing, you should use the structure `ArrayStockPackage`, which uses the `ArraySequence` implementation of `SEQUENCE`.

How to indicate parallel calls? As seen in recitation, you can use the function `par` (inside the structure `Primitives`) to express calls that will be run in parallel. Parallel operations can also be expressed in terms of operations on sequences such as `map` or `reduce`. *In your code, be explicit about what calls are being made in parallel.*

3.1 Naïve Implementation

Task 3.1 (5%).

Implement a quadratic work brute-force solution to the stock market problem in a functor called `StockNaive`.

3.2 Divide and Conquer

Task 3.2 (15%).

Implement a solution to the stock market problem by divide-and-conquer recursive programming. If the work of showing any tree view of a sequence is denoted W_{showt}, the work of your solution must be expressed by the recurrence

$$W_{stock}(n) = 2\left(W_{stock}\left(\frac{n}{2}\right)\right) + W_{showt} + O(1)$$

with

$$W_{stock}(0) \in O(1)$$

A solution with correct input-output behavior but with work that is not described by this recurrence will not receive full credit.[1] Write your implementation in a functor called `StockDivAndConq`.

Task 3.3 (15%).

Use mathematical induction on the length of the input sequence to prove that your divide and conquer implementation is correct. Be sure to carefully state the theorem that you're proving and to note all the steps in your proof. Remember that a longer proof is not necessarily a more correct proof.

Task 3.4 (10%).

The specification in Task 3.2 stated that the work of your solution must follow a recurrence that was parametric in the work it takes to view a sequence as a tree. Naturally, this changes with the sequence implementation.

1. Solve the recurrence with the assumption that $W_{showt} \in O\left(\lg n\right)$.
2. Solve the recurrence with the assumption that $W_{showt} \in O(n)$ where n is the length of the input sequence.
3. In two or three sentences, describe what data structure you would use to implement the sequence α `seq` so that `showt` would actually have $O(\lg n)$ work.
4. In two or three sentences, describe what data structure you would use to implement the sequence α `seq` so that `showt` would actually have $O(n)$ work.

3.3 Sequence Operations

Task 3.5 (10%).

Implement a solution to the stock market problem using `reduce` and possibly `map` rather than an explicit recursive function. Write your implementation in a functor called `StockReduce`.

Your function `StockReduce.stocks` *must not* call itself.

3.4 Testing Your Code

Task 3.6 (10%).

Write a functor called `TestStocks` that takes any structure ascribing to `STOCKS` ascribing and produces a structure ascribing to the signature `TESTS`. Your functor should thoroughly and carefully test the argument structure. This should include both edge cases and more general test cases on specific sequences.

Your naïve implementation will likely be simple enough to be obviously correct, so you can write more exhaustive tests by comparing the argument structure to the function against the naïve solution on many lists.

[1]*Hint:* You may want a helper function of type `int seq` $\rightarrow$ `int` $*$ `int` $*$ `int` or `int seq` $\rightarrow$ `int option` $*$ `int option` $*$ `int option`

4 Asymptotics

For this problem, let's recall the definition of big-O:

Definition 4.1. A function $f : \mathbb{N} \to \mathbb{R}_+$ is in $O(g)$ if and only if there exist constants $N_0 \in \mathbb{N}$ and $c \in \mathbb{R}_+$ such that for all $n \geq N_0$, $f(n) \leq c \cdot g(n)$.

Task 4.1 (5%). Rearrange the list of functions below so that it is ordered with respect to O—that is, for every index i, all of the functions with index less than i are in Big-O of the function at index i. You can just state the ordering; you don't need to prove anything.

1. $f(n) = n^{\lg \lg n}$
2. $f(n) = \lg(\lg(\lg(n)))$
3. $f(n) = n^{1.5}$
4. $f(n) = n!$
5. $f(n) = 2^{n!}$
6. $f(n) = 5^n$
7. $f(n) = n^7 + \frac{1}{2}n^2 + \sqrt{n}$

Task 4.2 (15%). Carefully **prove or disprove** each of the following statements. Remember that verbose proofs are not necessarily careful proofs: your answers for these questions will likely fit on one page.

1. O is a transitive relation on functions. That is to say, for any functions f, g, h, if $f \in O(g)$ and $g \in O(h)$, then $f \in O(h)$.
2. O is a symmetric relation on functions. That is to say, for any functions f and g, if $f \in O(g)$, then $g \in O(f)$.
3. O is an anti-symmetric relation on functions. That is to say, for any functions f and g, if $f \in O(g)$ and $g \in O(f)$, then $f = g$.

Task 4.3 (10%). Prove the following identities:

1. Show that for $a \in \mathbb{R}$, $a \neq 1$,
$$1 + a + a^2 + \cdots + a^n \; = \; \frac{a^{n+1} - 1}{a - 1}.$$
2. Show (by induction or otherwise) that for $n \geq 1$,
$$\sum_{i=1}^{n} i^3 = \left(\sum_{i=1}^{n} i \right)^2.$$