CS 104 (Fall 2013) — Assignment 4
Due: 10/03/2013, 11:59am

BitBucket directory name for this homework (case sensitive): HW4.

As of Homework 4, submissions via Blackboard will not be accepted any more. By now, you should have gotten your BitBucket account to work — if not, ask for help quickly.

Also, here is a clarification on how you should organize solutions to different problems, and how to use Makefiles: Immediately inside your homework folder, there has to be one and only one Makefile. Calling this Makefile should build your code and generate the right binary for each question. For example, calling `make q3` inside the HW4 folder should produce a `q3` binary immediately inside the HW4 folder. Running the `q3` binary should produce the required output for Question 3 of Homework 4. So it's your responsibility that all required files are in the right place for your code to run. Additionally, the Makefile should support the `clean` rule and the `all`. There are many Makefile tutorials on the web, in case you are still having trouble.

(1) Review C++ Interludes 1, 2, 4 (again) and Chapters 8, 9 from the textbook.

(2) In this question, you will experimentally evaluate different implementations of the `List` class from Chapters 8/9 in the textbook. As you will recall from class, we did not include all the functions that the textbook suggests, only the following four:

```
template <class T>
class List {
void insert (int pos, const T & item);
  /* Inserts the item right before position pos, growing the list by 1.
     pos must be between 0 and the current length of the list.
     (The textbook returns a bool - feel free to do that.) */
void remove (int pos);
  /* Removes the element at position pos, shrinking the list by 1.
     pos must be between 0 and the current length of the list minus 1. */
void set (int pos, const T & item);
  /* overwrites position pos in the list with item.
     Does not change the length of the list.
     pos must be between 0 and the current length of the list minus 1. */
T const & get (int pos) const;
  /* returns the item at position pos, not changing the list.
     pos must be between 0 and the current length of the list minus 1. */
};
```

It is up to you to decide what to do when `pos` is out of its legal range. You are asked to build four implementations of the `List` class and compare them experimentally. In doing so, you can certainly look at the textbook, which gives pretty much complete implementations. However, we strongly suggest that you attempt to solve this by yourself. (Also, a few things will be different, as you will see.)

(a) Implement a pure abstract version of the `List` class. (Hint: we almost completely solved that for you above.)

(b) Derive from `List` a class that builds an implementation based on linked lists. You may use your code from previous homeworks if it helps you.

(c) Derive from `List` three classes that each build implementations based on arrays. They will be identical, except for when they expand the dynamically allocated array. Remember that internally, you will store the items in an array. When you have inserted enough items to exceed the current size of the array, you allocate a larger array and copy over the elements. The three classes should have the following behavior:

- One of them increases the array size by 1 every time the current array is too small.
- One of them increases the array size by 10% (rounded up) every time the current array is too small.
- The last one doubles the array size every time the current array is too small.

(In all cases, watch out for memory leaks.)

For full credit (and good coding practice and reuse), create an intermediate abstract class that implements all functions except for one function called `void expandArray ();` or something like that, and just create the different classes by implementing these functions differently.

(d) For each of your four implementations, measure how long they take to insert n numbers into a `List<int>`. If you handled inheritance correctly in the previous subproblems, it should be trivial to change code from measuring one class to another.

Recall from class that inserting an element in position i takes $\Theta(i)$ for the Linked List, and $\Theta(n-i)$ for the Array. So to be fair, all elements should be inserted in the current middle of the list (plus/minus 1).

For values $n = 2000, 4000, 6000, 8000, 10000$, measure how long this insertion takes. In this case, do *not* read the numbers from a file, but rather just add the numbers from 1 to n (so we don't measure the time to read a file).

Once you have measured these times, use software to prepare a plot of the four curves for the different implementations. (We recommend gnuplot or Excel, but if you know Maple, Mathematica, or any other such software, feel free to use it.)

Also, report (in text) on your conclusion about the overhead incurred by the implementations compared to the fastest one.

(3) Suppose that you have a social network of n people. For each of these n people, you have a value $v_i \geq 0$ which tells you how much you would like to invite that person to your party. Unfortunately, there are also some pairs of people (enemies, exes) that you cannot simultaneously invite to your party, because they would fight with each other. What you want to do is find a group S of people you can invite such that no two people you invite will fight with each other, and which maximizes the total value to your party $\sum_{i \in S} v_i$ among all such safe groups of people.

Write a program that solves this problem for at least up to $n = 20$. You should read your input (network, values) from a file, but can choose the format in which things are stored in your file. As a recommendation, the following format may work quite well, though you are by no means required to use it:

- First line: number of people n, and number of pairs of enemies m.
- Second line: n floating point (or integer) values v_i. (Or you could have them one per line.)
- Next m lines: two integers each, describing one pair of enemies. (Or you could have n lines, and on each line i you list all the enemies of person i. In that case, you wouldn't even need to specify m on the first line.)

Example of the suggested format:

```
4 3
10 5.4 6.2 1
1 2
1 3
4 3
```

Almost certainly, you will want to use backtracking here. I would be surprised if your solutions works for $n > 40$, though it may work up to $n = 30$ or so.

(4) **Chocolate Problem: This problem does not count towards your grade, but if you solve it, you can earn up to 1 chocolate bar. Feel free to specify your preferred type of chocolate — your requests may be accommodated if they are reasonable. Submitting this problem a few days late is ok, but in that case, please submit it directly by e-mail to David.**

Write a Sudoku solver. If you don't know the rules of Sudoku, Google them (but don't get addicted to solving them). Find a suitable way to represent the input shape and given numbers to read from a file. Then use a technique of your choice to solve the puzzle and output the solution.