Big O Practice

CS 200 Recitation 4

One more thing about interfaces

Forgot to mention this last week, sorry.

- **Java interfaces cannot specify constructor methods!** This is because an interface can be implemented by multiple classes, and constructors are specific to a single class.
  - If you want to control class construction with an interface (maybe to require specific constructor arguments or something), you have to do it another way.
  - Methods which return an instance of the same class behave much like a constructor (especially if they are static). An interface can specify such a method as a way of controlling class construction.
  - **This is why there was a create() method in the PA1 skeleton files.**
  - Methods which exist to build class instances are often said to be using the *Factory Method design pattern.* Though strictly speaking, a factory method is a method which constructs a range of possible classes depending on the situation (often with the actual constructor declared as protected scope, so external uses must call the factory method).
  - I should also mention the *Singleton design pattern.* Where a Factory Method is used to maintain only a single instance of a class. The Factory Method will construct the class if no instance exists; If an instance exists, it will return that instance instead of building a new one. This can be used to store global state in a program.

Tip Of The Week

Handling the output of terminal commands, you can do more that just look at it on the screen. Two broad categories:

- **Redirect the output to a file.** Add "> filename" to the end of the command where filename is the name of a file (will be created if necessary). This example stores the output of ls in a file called dir_list.txt

  ```
  corn> ls > dir_list.txt
  ```

- **Pipes connect the output of one command to the input of another.** The "|" character is called a pipe when it is used in the terminal. Using pipes, you can chain several commands together. This example counts the number of things in a directory by calling ls and piping it's output to wc -l (word count, setup to count lines)

  ```
  corn> ls | wc -l
  52
  ```

More on Big O notation

Eyeballing code

The process of Looking at a piece of code and determining it's Big O complexity is more intuitive than mechanical. Generally you should look for where most of the work is being done, and/or for code that runs many times. Next you should think about how many times the code will run for input of a given size (n), and maybe trace out a small example. This should give you an idea of how to express the number of code cycles in terms of the size of it's input.

As an example, lets look at the *traveling salesman problem.* Finding the shortest route for a salesman to visit n cities. Each city is visited only once, and they can be visited in any order. Brute force computation of this problem is to compute all possible routes and then find the lowest distance. Adding up distance between cities is a constant time operation, so we will focus on enumerating all possible routes:

```
//arguments are two lists of cities
method trav_sales(visited, notVisited){
  if(notVisited is empty)
    return visited;
  list results;
  for(x is a city in notVisited){
    list temp = visited + x;
    list notV = remove x from notVisited;
    results.append(trav_sales(temp, notV)); //recursive call
  }
  return results;
}
```

Here we have a loop which contains a recursive call.

- Loop runs n times. After 1 recursion, loop runs n-1 times. After two recursions loop runs n-2 times, etc...
- First time through the loop, n recursive calls are made. Each one has a n-1 loop, so it is n-1 operations done n times. You can think of it as a nested loop (where the inner loop skips an element) if you want.
- So far it looks like $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$, which is n factorial (n!). This makes a bit of sense, since every time a city is visited, there are fewer cities which still need to be visited.
- I tried a few small inputs. For n=3, there were 6 routes (3! = 6). For n=4 there were 24 routes (4! = 24). So n! does seem to be a good representation of how much looping it does. Since it does not do much else, it's probably O(n!).
- I looked this one up, brute force solutions to the traveling salesman problem really are O(n!).

Proofs

Other problems ask us to prove that f(x) is O(g(x)), where f and g are mathematical functions. Remember the official definition for big O: $|f(x)| \leq C|g(x)|$, whenever $x > k$. I actually like the book's technique for this, so here is an example:

Show that $6x^2 + 2x + 3$ is $O(x^3)$

Start by writing down $f(x) \leq f(x)$. no, really. Stay with me, we get:

$$6x^2 + 2x + 3 \leq 6x^2 + 2x + 3$$

Now, we want to make the right hand side look as much like g(x) as we can. To do this, we make some observations. Specifically, one about each term in the original function, relating it to g(x):

$6x^2 \leq x^3$, for $x > 6$.

$2x \leq x^3$, for $x > 2$.
$3 \leq x^3$, for $x > 2$.

How did I find these? Plotted them in Wolfram|Alpha, looked for lowest integer where x^3 was bigger. That number is the bound on x above. Based on these observations, when x is greater than 6 every term in the original function is less than x^3. So we can rewrite the original in equality as:

$$6x^2 + 2x + 3 \leq 6x^3 + 2x^3 + 3x^3$$

I literally just made every term in the right side an x^3 term! In fact, we can remove the coefficients on the right side, the inequality still holds:

$$6x^2 + 2x + 3 \leq x^3 + x^3 + x^3$$
$$6x^2 + 2x + 3 \leq 3x^3$$

This now looks a lot like the definition, $|f(x)| \leq C|g(x)|$, with c = 3 and k = 6 (these are the witness variables). C from previous equations, and K is highest bound on x from our observations above. We can now say that:

$6x^2 + 2x + 3$ is $O(x^3)$ with witness: c=3 and k=6.

Exercise

Worksheet again, sorry.

- problems from the book (mostly Rosen book 6th ed, one from Prichard 3rd ed)
- I picked odd numbers again, so answers are in the back. I will also post answers to the overhead toward the end of the period and to RamCt after Thursday's recit.
- Grading: attendance and doing the work, just as before.
- When you are done, show me your worksheet and I will record your attendance. You may keep the worksheet.