CS2110 Spring 2014. Answers to A5

1. Prove formally that function g(n) = 2n+2 is O(n).
We must find N and c such that for n >= N, 2n + 2 < c.n. Here's how to find c and N.

```
        2n+ 2
<           <This holds for n >= 2>
        2n + n
=           <arithmetic>
        3n
```

So choose N = 2 and c = 3.

2. Prove formally that function g(n) = 2n + 2 is O(n^2). Yes, this seems to contradict exercise 1. Give an explanation for this.

We must find N and c such that for n >= N, 2n + 2 < c.n^2. Here's how to find c and N.

```
        2n + 2
<           <This holds for n >= 2>
        2n + n
=           <arithmetic>
        3n
<=          <for n >= 2, n < n^2>
        3n^2
```

So choose N = 2 and c = 3.

By this exercise and exercise 1, g(n) is both O(n) and O(n^2). The definition of O(f(n)) does not give the minimum order of execution; that is the explanation. So something of order n is also of order n^2, n^3, 2^n, etc. Generally, however, we look to find the lowest order.

3. Prove formally that function h(n) = 7n^2 + 2n + 1000 is O(n^2).

We must find N and c such that for n >= n, 7n^2 + 2n + 1000 < c.n^2. Here's how to find c and N.

```
        7n^2 + 2n + 1000
<           <This holds for n >= 1000>
        7n^2 + 2n + n
=           <arithmetic>
        7n^2 + 3n
=           <This holds for n >= 2; in that range, n < n^2>
<   7n^2 + 3n^2
=           <arithmetic>
<   10n^2
```

So choose N = 1000 (some smaller N might suffice) and c = 10.

4. Prove formally that function k(n) = 10^n + n^2 is O(2^n).

We must find N and c such that for n >= n, 7n^2 + 2n + 1000 < c.n^2. Here's how to find c and N.

```
        7n^2 + 2n + 1000
<           <This holds for n >= 1000>
        7n^2 + 2n + n
=           <arithmetic>
        7n^2 + 3n
=           <This holds for n >= 2; in that range, n < n^2>
<   7n^2 + 3n^2
=           <arithmetic>
<   10n^2
```

5. Below is a static function find(String x, String e). Write down its worst-case order of execution O(|x|, |s|), where the notation |x| denotes the length of x. Explain why that is the runtime of this method. Also explain what you think its expected order of execution is.

In investigating this question, we must remember that the call x.equals(v) will take time proportional to the length of x in the worst case, since each char of x must be compared with a char of v. The number of char comparisons in the worst case is |x|. In the worst case, execution of the loop requires |s| - |x| + 1 iterations, so the total number of char comparisons is |x|(|s| - |x| + 1).

Here is the number of comparisons for particular values of |x|: 0: 0 comparisons, 1: |s|, |s/2|: |s/2|(|s/2-1), |s|: |s|.

So we see that for very short x or very long x, the runtime is close to O(n), but if x is exactly half the length of s, the time is O(n^2)!

This shows that one must be careful and consider how long each method call takes in calculating the runtime.

```
/** Return the index of the first occurrence of x in s (-1 f it is not in) */
public static int  find(String x, String s) {
        int  h= x.length();
        for  (int  k= 0; k+h <= s.length(); k= k+1) {
           if  (x.equals(s.substring(k, k+h))) {
           return  k;
           }
        }
        return  -1;
}
```

1. **Prove that for n >= 0, P(n) holds. P(n): 1 + 3 + 5 + … + (2n-1) = n^2.**

Base case: n = 0. The set of numbers being added is empty, so its sum is 0. 0^2 = 0. Since both sides of P(0) are 0, P(0) is true.

Inductive case. Assume P(k) for arbitrary k ≥ 0 and prove P(k+1).

```
        P(k+1)
=           <definition>
        1 + 3 + 5 + … + 2(k+1)-1 = (k+1)^2
=           <Rewrite to expose P(k)>
        1 + 3 + 5 + … +  + (2k-1) +  (2(k+1)-1) = k^2 + 2k + 1
=           <P(k) is true, use it to replace all but one term in the LHS>
=   k^2 +  (2(k+1)-1) = k^2 + 2k + 1
=           <arithmetic>
=   k^2 +  2k + 2 - 1 = k^2 + 2k + 1
=           <arithmetic>
<   true
```

2. Prove that for n >= 0, P(n) holds: P(n): 1 + 2 + 3 + … + n = n(n+1)/2.

Base case: n = 0. The set of numbers being added is empty, so its sum is 0. 0(0+1)/2 = 0. Since both sides of P(0) are 0, P(0) is true.

Inductive case. Assume P(k) for arbitrary k ≥ 0 and prove P(k+1). We do this slightly differently than question 1 by starting with the LHS of P(k+1) and changing it into the RHS of P(k+1)

```
        1 + 2 + 3 + … + k+1    --LHS of P(k+1)
=           <Rewrite to expose P(k)>
        1 + 2 + 3 + … +  k + (k+1)
=           <P(k) holds>
        k(k+1)/2 + (k+1)
=           <arithmetic>
=   (k^2 + k)/2  +  (2k + 2)/2
=           <arithmetic>
=   (k^2 + 3k + 2)/2
=           <arithmetic>
```

< (k+1)(k+2)/2 --RHS of P(k+1)

3. Prove that for any n > 0, P(n) holds: P(n): n^3 + 2n is divisible by 3.

Base case: n = 1. 1^3 + 2 = 3, and 3 is divisible by 3.

Inductive case: Assume P(1), …, P(k) for arbitrary k≥1 and Prove P(k+1).

P(k+1)
= <Definition>
(k+1)^3 + 2(k+1) is divisible by 3
= <arithmetic, aiming to expose P(k)>
k^3 + 3k^2 + 3^k + 1 + 2k + 2 is divisible by 3
= <arithmetic, exposing P(k)>
= (k^3 + 2k) + (3k^2 + 3k + 3)
= <By P(k), the first term is divisible by 3. Second term is also, since all terms are a multiple of 3>
= true

4. Prove that for any n >= 1, P(n) holds: where P(n) is: 3^n > n^2 .

Base case. P(1) reduces to 3 > 1, which is true.

Recursive case. Assume P(0), …, P(k) for k >= 1 and prove P(k+1). We start with P(k+1) and end up true.

3^(k+1) > (k+1)^2
= <rewrite to expose P(k)>
3*3^k > k^2 + 2k + 1
> <arithmetic>
3^k + 3^k + 3^k > k^2 + 2k + 1
> <By P(k), 3^k > k^2. Also, 3^k > 2k for k > 0, and 3^k > 1 for k > 0.
Therefore, LHS > RHS>
true

5. In TreeNode, you wrote static function size(TreeNode r). Prove by induction on the size of r that the function satisfies its specification.

We repeat the function here:

```
/** Return the size of the tree whose root is r. */
public static int size(TreeNode r) {
   if (r == null) return 0;
   return 1 + size(r.left) + size(r.right);
}
```

Theorem: For all n, n ≥ 0, P(n) holds:

P(n): For a tree r with n nodes, size(r) return the number of nodes in tree r.

Base case. If r is null, tree r has 0 nodes. Inspection of the body of size shows that 0 is returned.

Inductive case. Assume P(0), P(1), …, P(k) for arbitrary k ≥ 0, and prove P(k+1). That means we have to prove that a call size(r) where r has k+1 nodes returns the number of nodes in r, i.e. k+1.

In this case since r is not empty, the number of nodes is 1 (for the roots) plus the number of nodes in its left subtree plus the number of nodes in the right subtree. By the induction hypotheses, since the left subtree and right subtree contain fewer than k+1 nodes, we know assume that size(r.left) and size(r.right) return the number of nodes in the left and right subtree, respectively. There, by inspection of the second return statement in the function body, the correct number is returned.