1 Algebra and Theta

Each of the complex functions on the left can be transformed algebraically until it equals (or is a constant multiple) of one of the functions on the right. This means, of course, what the function on the left is Theta of the function on the right.

For each function on the left, transform it until it is obvious that it is Theta of some function on the right (not necessarily the one it’s next to); then state the equivalence as a Theta expression.

**EXAMPLE:**

\[ \lg \sqrt{n} = \lg n^{1/2} = \frac{1}{2} \lg n = \Theta(\lg n) \]

<table>
<thead>
<tr>
<th>LEFT</th>
<th>RIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lg n^2 )</td>
<td>( \lg n )</td>
</tr>
<tr>
<td>( 4 \lg n )</td>
<td>( \sqrt{n} )</td>
</tr>
<tr>
<td>( \lg n^n )</td>
<td>( n )</td>
</tr>
<tr>
<td>( 4^{\lg n} )</td>
<td>( n \lg n )</td>
</tr>
<tr>
<td>( 2^{\lg n} )</td>
<td>( n^2 )</td>
</tr>
<tr>
<td></td>
<td>( 2^n )</td>
</tr>
</tbody>
</table>

**Solution:**

\[ \lg n^2 = 2 \lg n = \Theta(\lg n) \]
\[ 4 \lg n = \Theta(\lg n) \]
\[ \lg n^n = n \lg n = \Theta(n \lg n) \]
\[ 4^{\lg n} = 2^{2 \lg n} = 2^{\lg n^2} = n^2 = \Theta(n^2) \]
\[ 2^{\lg n} = n = \Theta(n) \]

2 Ordering of Functions

Justify each of the following statements. Formal proofs are not required; simply make an argument from the rules we’ve discussed in class. Some algebraic transformations may be required.

(a) \[ \lg n = o(\sqrt{n}) \]

**Solution:** In class, we stated that any logarithm grows strictly slower than any (nonzero) polynomial function.
(b) \[ \sqrt{n} = o(n) \]

**Solution:** \( \sqrt{n} = n^{1/2} \). In class, we stated that any small polynomial power grows strictly slower than any larger polynomial power.

(c) \[ n = o(n \lg n) \]

**Solution:** \( n \) grows more slowly than \( n \lg n \) because the latter has the extra \( \lg n \) term. Thus, there is no constant we could use to make \( n \) keep up with \( n \lg n \); eventually, the latter would get much, much larger.

(d) \[ n \lg n = o(n^2) \]

**Solution:** In class, we stated that any logarithm grows strictly slower than any polynomial. Thus, while \( n \lg n \) grows faster than \( n \), it grows much, much less quickly than \( n^2 \).

## 3 Some Properties of Asymptotic Notation

Prove or disprove the following conjectures about asymptotic notation (use the formal definitions of asymptotic notation when appropriate):

(a) \[ f(n) + g(n) = \Theta(\min(f(n), g(n))) \]

**Solution:**
This conjecture is false.
Consider the example of \( f(n) = n, g(n) = n^2 \). Clearly, (for large \( n \)) \( \min(f(n), g(n)) \) is approximately \( n \), whereas the function \( f(n) + g(n) \) is approximately \( n^2 \). Thus, the conjecture is false.

(b) If \( f_1(n) = O(g_1(n)) \) and \( f_2(n) = O(g_2(n)) \), then \( f_1(n) + f_2(n) = O(g_1(n) + g_2(n)) \).

**Solution:**
This conjecture is true.
Because \( f_1(n) = O(g_1(n)) \), we know that there exists some \( c_1 \) such that, for sufficiently large \( n \), \( f_1(n) \leq c_1 g_1(n) \). The same is also true for \( f_2(n), g_2(n) \).
Now, let us choose \( c_3 = \max(c_1, c_2) \). We can easily see that (for large \( n \))
\[ f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n) \leq c_3 g_1(n) + g_2(n) \]
Therefore, \( f_1(n) + f_2(n) = O(g_1(n) + g_2(n)) \).

(c) If \( f_1(n) = O(g_1(n)) \) and \( f_2(n) = O(g_2(n)) \), then \( f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n)) \).

**Solution:**
This conjecture is **true**.
Because \( f_1(n) = O(g_1(n)) \), we know that there exists some \( c_1 \) such that, for sufficiently large \( n \), \( f_1(n) \leq c_1 g_1(n) \). The same is also true for \( f_2(n), g_2(n) \).
We can easily see that
\[
f_1(n) \cdot f_2(n) \leq c_1 g_1(n) \cdot c_2 g_2(n) \leq c_1 c_2 (g_1(n) \cdot g_2(n))
\]
Therefore, \( f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n)) \).

(d) if \( f(n) = \omega(g(n)) \) then \( f(n) + g(n) = O(f(n)) \).

**Solution:**
This conjecture is **true**.
By the definition of \( \omega \), we know that, for any possible \( c > 0 \), for sufficiently large \( n \), \( c g(n) < f(n) \).
Therefore, we can conclude that (for any \( c > 0 \), and for large \( n \)):
\[
f(n) + g(n) < f(n) + \frac{f(n)}{c} = (1 + \frac{1}{c})f(n)
\]
Therefore
\[
f(n) + g(n) = O(f(n))
\]

4 Tuples - Storing and Comparing

Programs commonly store tuples in various forms (although very often, they don’t use the technical term); a tuple is a data structure with a fixed number of fields, each of which has a certain type. The order of these fields matter, as each one normally represents a certain piece of data about an object. (See the Review deck.)

(a) Declare a C struct or Java class which models a tuple with 4 fields; be clear which language you are using. (If you write it in Java, do not include any methods, and all of the data fields should be public.)

**Solution:**
Java:

```java
public class Tuple4 {
```

Page 3
int id;
int age;
String lastName;
String firstName;
}

C:

struct Tuple4
{
    int id;
    int age;
    char* lastName;
    char* firstName;
}

(b)

If you wrote your tuple in Java, write a member method for the class above which takes a single parameter (another object of the same type). This method should work like `String.compareTo()`: it returns negative if the `this` object should come earlier in a sorted order, and positive if the parameter should come earlier; it returns 0 only if the two tuples are equal.

(Make sure that your method takes into account all of the variables; it should only return zero if all of the fields are identical.)

If you wrote your tuple in C, do the equivalent: write a function which takes two pointers to your tuple objects, compares them, and returns negative, positive, or zero based on how the two should be arranged. (In C, a comparable function is `strcmp`.)

Solution:
Java:

    public int compareTo(Tuple4 other)
    {
        if (this.id != other.id)
            return this.id - other.id;

        if (this.age != other.age)
            return this.age - other.age;

        if (this.lastName.equals(other.lastName) == false)
            return this.lastName.compareTo(other.lastName);

        return this.firstName.compareTo(other.firstName);
    }

C:

    int compareTo(Tuple4* x, Tuple* y)
{  
  if (x->id != y->id)
      return x->id - y->id;

  if (x->age != other->age)
      return x->age - y->age;

  if (strcmp(x->lastName, y->lastName) != 0)
      return strcmp(x->lastName, y->lastName);

  return strcmp(x->firstName, y->firstName);
}