COMP 250: Introduction to Computer Science
Assignment 2 - Sample solutions

1. [20 points] **Proofs by induction**

   (a) Prove by induction on n the formula for the geometric series: for any natural numbers $b > 1$, $n > 1$,

   $$\sum_{i=0}^{n} b^i = \frac{b^{n+1} - 1}{b - 1}$$

   Solution:
   Base case: For $n = 0$, $b^0 = \frac{b-1}{b-1} = 1$
   Induction step:

   $$\sum_{i=0}^{n+1} b^i = \sum_{i=0}^{n} b^i + b^{n+1} = \frac{b^{n+1} - 1}{b - 1} + b^{n+1} = \frac{b^{n+1} - 1 + b^{n+2} - b^{n+1}}{b - 1} = \frac{b^{n+2} - 1}{b - 1}$$

   where we used the induction hypothesis in the second equality.

   (b) Prove by induction that for all positive integers n:

   $$\sum_{k=1}^{n} \frac{1}{k(k+1)} = \frac{n}{n+1}$$

   Solution:
   Base case: For $n = 1$, $\frac{1}{1*2} = \frac{1}{2}$
   Induction step:

   $$\sum_{k=1}^{n+1} \frac{1}{k(k+1)} = \sum_{k=1}^{n} \frac{1}{k(k+1)} + \frac{1}{(n+1)(n+2)} = \frac{n}{n+1} + \frac{1}{(n+1)(n+2)} = \frac{n^2 + 2n + 1}{(n+1)(n+2)} = \frac{n+1}{n+2}$$

   where we used the induction hypothesis in the second equality.

   (c) Prove by induction that $n^2 < 2^n, \forall n > 4$
   Solution:
   Base case: For $n = 5$, $5^2 = 25 < 2^5 = 32$ is true.
   Induction step:

   $$(n+1)^2 = n^2 + 2n + 1 < 2^n + 2n + 1$$

   by the induction hypothesis. Note that $2n + 1 < n^2$, because $n^2 - 2n - 1 = (n-1)^2 - 2$ which is positive for $n > 4$. Hence:

   $$2n + 1 < n^2 < 2^n$$

using the induction hypothesis again. Putitng these together we get:

$$(n+1)^2 < 2^n + 2^n = 2^{n+1}$$

which concludes the proof.

(d) Prove by induction that $8^n - 2^n$ is divisible by 6.

Solution:

Base case: For $n = 1$ we have $8^1 - 2^1 = 6$ which is divisible by 6.

Induction step: $8^{n+1} - 2^{n+1} = 8^n * 8 - 2^n * 2 = 8^n * 6 + 2 * (8^n - 2^n)$. Since the first term in the sum is divisible by 6 and the second term is divisible by 6 from the induction hypothesis, the sum is divisible by 6, which concludes the proof.

2. [20 points] **Big-Oh**

(a) Prove that $10000n + 10^6 \in O(n)$.

Solution: Apply the definition of $O()$ with $n_0 = 10^6$, $c = 10000$.

(b) Prove that 3^n is not $O(n^3)$.

Solution: Suppose by contradiction that $3^n \in n^3$, so there must exist c, n_0 such that $3^n < cn^3, \forall n > n_0$, which equivalently means $n \log_3 3 < \log_3 c + 3 \log_3 n$, or $n < \log_3 c + 3 \log_3 n$, so $0 < \log_3 c + 3 \log_3 n - n$. The limit of the right-hand-side is $-\infty$, so the statement cannot be true.

An alternative proof can be given using the limits rule and applying l'Hopital's rule repeatedly.

(c) Is $O(n \log_2 n)$ in $O(n^2)$? Prove your answer

Solution: Yes. Consider any function $f \in O(n \log_2 n)$, so there exist constants c, n_0 such that $f(n) < cn \log_2 n < cn \cdot n = n^2, \forall n < n_0$, so $f \in O(n^2)$ as well.

(d) Give an example of two functions f and g such that $f \notin O(g)$ and $g \notin O(f)$.

Solution: We need two function such that neither dominates the other all the time. To ensure this, it is natural to think of periodic functions. For example, consider the tangent and cotangent functions - they are periodic and each dominates the other over an infinite set of periodic intervals. Many others are possible as well.

3. [15 points] **More Big-oh**

For the following pieces of code, give the tightest $O()$ estimate that you can, and justify your answer.

(a) int $sum = 0$;
for (int $i = 0; i < n; i = i + 2$);

for (int $j = 0; j < 10; j++$)
$sum = sum + i + j$;

Solution: The inner loop executes a constant number of times (10), the outer loop executes $n/2$ times, which is $O(n)$, so overall we get $O(n)$.

(b) int $sum = 0$;
for (int $i = n; i > n/2; i--$);
for (int $j = 0; j < n; j++$)
$sum = sum + i + j$;

Solution: The inner loop is $O(n)$, the outer loop is also $O(n)$ (we got to $n/2$ decreasing by a fixed amount every time) and since the loops are nested, we get $O(n^2)$ for the whole code.

(c) int $sum = 0$;
for (int $i = n; i > n - 2; i--$);
for (int $j = 0; j < n; j+ = 5$)
$sum = sum + i + j$;

Solution: The outer loop only executes twice, so it is $O(1)$, while the inner loop is $O(n)$, so the nested loops give $O(n)$

4. [25 points] **Recursion**

Write, in Java, a recursive method countBinaryStrings that has one integer parameter n and returns the number of binary strings of length n that do not have two consecutive 0's. For example, for $n = 4$, the number of binary strings of length 4 that do not contain two consecutive 0's is 8: 1111, 1110, 1101, 1011, 1010, 0111, 0110, 0101. For this problem, your method needs to return *only the number of such strings*, not the strings themselves. You may assume that the integer specified in the parameter is positive. Looking at the example above will give you a hint about how such strings start.

The method should be static and embedded in a class called Recursion. This class should also have a main method. In this case, we will call the main method with an argument, the number of bits n. This argument will be in args[0]. You should convert it to an int using the Integer.parseInt method. Look this method up in the Java documentation to see what it does.

Solution: The main idea is to notice the structure of the recursion. For all such strings, either we have a 1 in the first position, and then we count the number of strings of length $n - 1$ with the property, or if we have a 0 in the first position, we need it to be followed by a 1, and then we count the strings with length $n - 2$. The code is provided in Recursion.java

5. [20 points] **More recursion**

Suppose we want to compute an exponential function b^n (where b is some base and n is an integer. There is a simple $O(n)$ algorithm for this (multiply b by itself n times). However, in this question you would have to devise a faster algorithm.

(a) [10 points] Devise an algorithm for solving this problem that works in $O(\log_2(n))$.

Solution: We will write this recursively. The main idea is that for even n, we can compute $b^{n/2}$ once then multiply the result together. For odd n, we will multiply b by the recursive call, which will be on an even argument.

Algorithm FastExp(b,n)
Input: a real number b and a positive integer n
Output: b^n

if $(n = 0)$ **then return** 1
double res =FastExp(b, $n/2$) // Note that integer division rounds down
$res = res * res$
if (n **mod** $2 \neq 0$) **then** $res = res * b$ //n is odd, so we need to multiply one more time
return res

(b) [10 points] Prove by induction that your algorithm works correctly.

Solution:

Base case: For $n = 0$ the algorithm returns correctly 1.

Induction step: Suppose n is odd, so $n = 2k + 1$ for some $k \geq 0$. By induction hypothesis, FastExp(b, k) works correctly and computes b^k, in which case our algorithm will return $b^k * b * k * b = b^{2k+1} = b^n$. If n is even, this means $n = 2k$ for some $k \geq 1$. By induction hypothesis, FastExp(b, k) works correctly and computes b^k, in which case our algorithm will return $b^k * b * k = b^{2k} = b^n$. This concludes the proof.