COMS 311: Homework 1
Due: Feb 9^{th}, 11:59pm
Total Points: 30 (+10 extra credit)

Late submission policy. Any assignment submission that is late by not more than two business days from the deadline will be accepted with 20% penalty for each business day. That is, if a homework is due on Friday at 11:59 PM, then a Monday submission gets 20% penalty and a Tuesday submission gets another 20% penalty. After Tuesday no late submissions are accepted.

Submission format. Homework solutions will have to be typed. You can use word, LaTeX, or any other type-setting tool to type your solution. Your submission file should be in pdf format. Do **NOT** submit a photocopy of handwritten homework except for diagrams that can be hand-drawn and scanned. We reserve the right **NOT** to grade homework that does not follow the formatting requirements. Name your submission file: `<Your-net-id>-311-hw1.pdf`. For instance, if your netid is `asterix`, then your submission file will be named `asterix-311-hw1.pdf`. Each student must hand in their own assignment. If you discussed the homework or solutions with others, a list of collaborators must be included with each submission. Each of the collaborators has to write the solutions in their own words (copies are not allowed).

General Requirements

- When proofs are required, do your best to make them both clear and rigorous.
- Even when proofs are not required, you should justify your answers and explain your work.
- When asked to present a construction, you should show the correctness of the construction.

Some Useful (in)equalities

- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $2^{\log_2 n} = n$, $a^{\log_b n} = n^{\log_b a}$, $n^{n/2} \le n! \le n^n$, $\log x^a = a \log x$
- $\log(a \times b) = \log a + \log b$, $\log(a/b) = \log a - \log b$
- $a + ar + ar^2 + ... + ar^{n-1} = \frac{a(r^n - 1)}{r-1}$
- $1 + \frac{1}{2} + \frac{1}{2^2} + ... + \frac{1}{2^n} = 2(1 - \frac{1}{2^{n+1}})$
- $1 + 2 + 4 + ... + 2^n = 2^{n+1} - 1$

Problems

1. **(5 pts)** Prove $\left[\frac{n(n+1)}{2}\right]^2 - \frac{n^2(n^2+1)}{4} + 78 \in O(n^3)$.

   Let's start by rewriting the LHS to simplify it.

$$\left[\frac{n(n+1)}{2}\right]^2 - \frac{n^2(n^2+1)}{4} + 78$$
$$= \left[\frac{n^2+n}{2}\right]^2 - \frac{n^4+n^2}{4} + 78$$
$$= \left[\frac{(n^2+n)(n^2+n)}{2^2}\right] - \frac{n^4+n^2}{4} + 78$$
$$= \left[\frac{n^4+n^3+n^3+n^2}{4}\right] - \frac{n^4+n^2}{4} + 78$$
$$= \frac{n^4}{4} + \frac{2n^3}{4} + \frac{n^2}{4} - \frac{n^4}{4} - \frac{n^2}{4} + 78$$
$$= \frac{n^4}{4} - \frac{n^4}{4} + \frac{2n^3}{4} + \frac{n^2}{4} - \frac{n^2}{4} + 78$$
$$= \frac{2n^3}{4} + 78 = \frac{n^3}{2} + 78$$

   Now that we have a simpler equation, $\frac{n^3}{2} + 78$, we can show that this function $\in O(n^3)$.

   The definition of $f(n) \in O(g(n)$ is: $\exists c, n_0$, such that $\forall n \geq n_0, f(n) \leq c \cdot g(n)$.

   For this problem, $f(n) = \frac{n^3}{2} + 78$ and $g(n) = n^3$.

   Since $f(n) \leq c \cdot g(n)$, we can alter f(n) to something that is definitively greater, so if $f'(n) \leq c \cdot g(n)$, then $f(n) \leq c \cdot g(n)$.

$$f(n) = \frac{n^3}{2} + 78, \qquad f'(n) = \frac{n^3}{2} + 78n^3 = f'(n) = \frac{n^3}{2} + \frac{156n^3}{2} = \frac{157n^3}{2}$$

   $f(n) \leq f'(n)$. If we choose c = 79 and $n_0 = 0$, our functions will now fit the definition.

   $f'(n) = 78.5n^3$, $\qquad g(n) = n^3$, c = 79.

   $\forall n \geq n_0$, $78.5n^3 \leq 79 \cdot n^3$

   $\forall n \geq n_0$, $f'(n) \leq c \cdot g(n)$, therefore $f(n) \leq c \cdot g(n)$, therefore $f(n) \in O(g(n))$.

   After this proof, we can conclusively say: $\left[\frac{n(n+1)}{2}\right]^2 - \frac{n^2(n^2+1)}{4} + 78 \in O(n^3)$.

2. **(5 pts)** Prove or disprove $2^{2^n} \in O(2^{2n})$.

$2^{2^n} \leq c \cdot 2^{2n}$
$2^n \cdot log2 \leq logc + 2n \cdot 2$
Remove both log 2
$2^n \leq logc + 2n$
Subtract 2n over
$2^n - 2n \leq logc$

This inequality is false. The constant cannot be greater than variable n. Since the inequality is not true, $2^{2^n} \notin O(2^{2n})$ because we cannot multiply the RHS by a constant to make it greater than the LHS, contradicting the definition of being a member of a big O set.

3. **(5 pts)** Prove that any function that is in $O(\log_2 n)$ is also in $O(\log_3 n)$.

   If $f(n) \in O(\log_2 n)$, then $f(n) \in O(\log_3 n)$.

   $\log_2 n = \frac{\log_3 n}{\log_3 2}$ - this is the change of base formula.

   The denominator, $\log_3 2$, is a constant. In runtime evaluation, we remove constants. Therefore, the denominator can be removed, leaving:

   $\log_2 n = \log_3 n$

   Hence, any function that is in $O(\log_2 n)$ is also in $O(\log_3 n)$.

4. **(5 pts)** Prove that if $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then

$$\boldsymbol{f_1(n) + f_2(n) \in O(g_1(n) + g_2(n))}$$

For $f_1(n) \in O(g_1(n))$ to hold, $f_1(n) \leq c \cdot g_1(n)$ for some constant c. Therefore, $f_1(n) \leq c \cdot g_1(n)$ is true.

For $f_2(n) \in O(g_2(n))$ to hold, $f_2(n) \leq c \cdot g_2(n)$ for some constant c. Therefore, $f_2(n) \leq c \cdot g_2(n)$ is true.

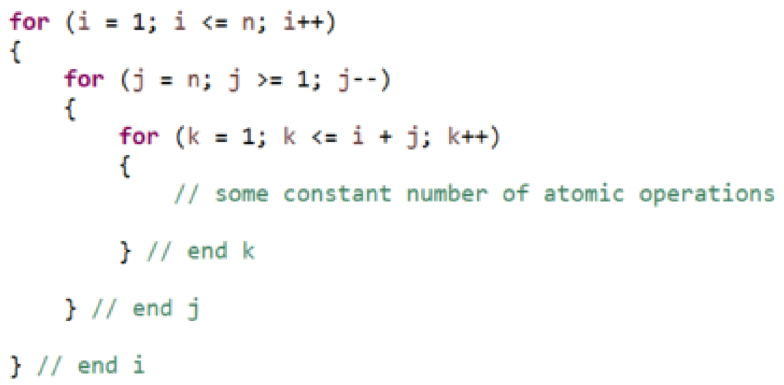
Therefore, $O(g_1(n) + g_2(n)) = c \cdot g_1(n) + c \cdot g_2(n) = c \cdot (g_1(n) + g_2(n))$.

For $f_1(n) + f_2(n) \in O(g_1(n) + g_2(n))$ to hold, $f_1(n) + f_2(n) \leq c \cdot (g_1(n) + g_2(n))$ must be true.

If $f_1(n) \leq c \cdot g_1(n)$ is true and $f_2(n) \leq c \cdot g_2(n)$ is true, $f_1(n) + f_2(n) \leq c \cdot (g_1(n) + g_2(n))$.

Hence, $f_1(n) + f_2(n) \in O(g_1(n) + g_2(n))$ must be true because $f_1(n) + f_2(n) \leq c \cdot (g_1(n) + g_2(n))$, which is the definition of $\in O$.

5. **(10 pts)** Derive the runtime of the following loop structure as a function of n and determine its Big-O upper bound. You must show the derivation of the end result. Simply stating the final answer without any derivation steps will result in zero points. Assume atomic operations take unit time.



```
for (i = 1; i <= n; i++)
{
    for (j = n; j >= 1; j--)
    {
        for (k = 1; k <= i + j; k++)
        {
            // some constant number of atomic operations

        } // end k

    } // end j

} // end i
```

For this problem, let's work inside out.
The line: for(k = 1; k $\leq$ i + j; k++) $\rightarrow$ This is equal to $\sum_{k=1}^{i+j} c_1$, with c being a constant number of operations.
The line: for(j = n; j $\geq$ 1; j–) $\rightarrow$ Although the loop counts down, this line is equal to $\sum_{j=1}^{n} c_2$, because counting down from n and up to n are the same. c_2 is the operations in the inner loop.
The line: for(i = 1; i $\leq$ n; i++) $\rightarrow$ This line is equal to $\sum_{i=1}^{n} c_3$, with c_3 being the number of operations inside the body of the loop.

This leaves us with: $\sum_{i=1}^{n} c_3$, $c_3 = \sum_{j=1}^{n} c_2$, $c_2 = \sum_{k=1}^{i+j} c_1$, $c_1 = 1$.

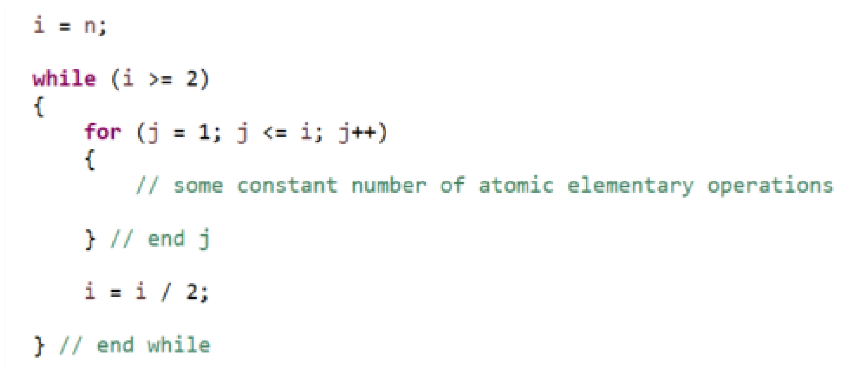
All in all, we get: $\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{i+j} c_1$
$\sum_{k=1}^{i+j} c_1 = c_1(\text{i} + \text{j})$
$\sum_{j=1}^{n} c_1(i+j) = c_1 \cdot \sum_{j=1}^{n} i + c_1 \cdot \sum_{j=1}^{n} j = i * n \cdot c_2 + c_2 \cdot \frac{n(n+1)}{2}$
$\sum_{i=1}^{n} i * n \cdot c_2 + c_2 \cdot \frac{n(n+1)}{2} = c_2 \cdot \frac{n(n+1)}{2} + c_2 \cdot \frac{n(n+1)}{2} = c_3 \cdot n^2(n+1) = c_3 \cdot (n^3 + n^2)$
c_1, c_2, c_3 are constants representing some constant number of operations.

The definition of being a member of a big O set is: $\exists c, n_0$, such that $\forall n \geq n_0, f(n) \leq c \cdot g(n)$.

Since $n^3 + n^2 \leq n^3 + n^3$ and $n^3 \leq n^3$ both hold, the definition hold true.

Since this definition holds true for c = 1 and $n_0 = 0$, $n^3 \leq \cdot n^3$, **this code has a big O runtime of:** $O(n^3)$.

6. **Extra Credit (10 pts)** Derive the runtime of the following loop structure as a function of n and determine its Big-O upper bound. You must show the derivation of the end result. Simply stating the final answer without any derivation steps will result in zero points. Assume atomic operations take unit time.



```
i = n;

while (i >= 2)
{
    for (j = 1; j <= i; j++)
    {
        // some constant number of atomic elementary operations

    } // end j

    i = i / 2;

} // end while
```

The runtime of this code is O(logn). It is obvious that the outer while loop will take logn time because i is divided by two each time. However, some may think that the inner loop will take n time because it runs for each iteration of the while loop. This is true, but the inner loop only goes to i, so it will run less and less times as the program goes on. The outer loop will dominate the time complexity, making the inner loop essentially an added constant because they decrease at the same rate.

In conclusion, the runtime will be O(logn) because the outer loop and inner loop decrease at the same rate.