COMS 311: Homework 1
Due: Sept 12, 11:59pm
Total Points: 40

Late submission policy. If you submit by Sept 13, 11:59PM (and after the specified deadline), there will be 10% penalty. That is, if your score is x points, then your final score for this homework after the penalty will be $0.9 \times x$. Similarly, if you submit by Sept 14, 11:59PM (and after Sept 13, 11:59PM), there will be 20% penalty.

Submission after Sept 14, 11:59PM will not be graded without (prior) explicit permission from the instructors.

Submission format. Your submission file should be in pdf format. 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`.

You are required to type-set your solution. You can use word, latex and any other type-setting tool to type your solution; you need to make sure you export/save it in pdf before submission.

Learning outcomes.

1. Determine whether or not a function is Big-O of another function
2. Analyze asymptotic worst-case time complexity of algorithms

Some Useful (in)equalities.

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

---

1. Prove or disprove the following statements. (20 Points)

   (a) $\left(\frac{n(n+1)}{2}\right)^2 - \frac{n^2(n^2+1)}{4} + 78 \in O(n^3)$.

   $\left(\frac{n(n+1)}{2}\right)^2 - \frac{n^2(n^2+1)}{4} + 78 \leq cn^3$

   $\frac{n^4+2n^3+n^2}{4} - \frac{n^4+n^2}{4} + 78 \leq cn^3$

   $\frac{n^3}{2} + 78 \leq cn^3$

   Therefore, **there exists** a c such that $c = 79$ **there exists** a n_0 such that $n_0 \geq 1$ where $\forall n \geq n_0 : \left(\frac{n(n+1)}{2}\right)^2 - \frac{n^2(n^2+1)}{4} + 78 \leq c \times O(n^3)$.

   (b) $2^{2^n} \in O(2^{2n})$

   $2^{2^n} \leq c2^{2n}$

   $2^n \leq \log_2 c + 2n$

   $n \leq \log_2(\log_2 c) + 1 + \log_2 n$

   $n \leq \log_2 n$ is a contradiction.

   Therefore, $2^{2^n} \notin O(2^{2n})$

   (c) Any function that is in $O(\log_2 n)$ is also in $O(\log_3 n)$.

   The statement above is true if $\log_2 n \in O(\log_3 n)$, so let's prove that...

   $\log_2 n \leq c \log_3 n$

   $\log_2 n = \frac{log_3 n}{log_3 2}$

   $\frac{log_3 n}{log_3 2} \leq c \log_3 n$

   Therefore, **there exists** a c such that $c = \frac{1}{\log_3 2}$ **there exists** a n_0 such that $n_0 \geq 1$ where $\forall n \geq n_0 : \log_2 n \leq c \times \log_3 n$.

   (d) If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then $f_1(n) + f_2(n) \in O(g_1(n) + g_2(n))$.

   What the first part is saying is that $f_1(n) \leq c_1 \times g_1(n)$ AND $f_2(n) \leq c_2 \times g_2(n)$.

   We are trying to prove that $f_1(n) + f_2(n) \leq c \times (g_1(n) + g_2(n))$.

   Since both $f_1(n) \leq c_1 \times g_1(n)$ AND $f_2(n) \leq c_2 \times g_2(n)$, we can intuitively see that the inequality we are trying to prove is true. However, it only holds true when $c =$ the greater of c_1 and c_2.

   Therefore, **there exists** a c such that $c = c_1$ OR c_2 (whichever is greater) **there exists** a n_0 such that $n_0 \geq 1$ where $\forall n \geq n_0 : f_1(n) + f_2(n) \leq c \times (g_1(n) + g_2(n))$.

2. Derive the runtime of the following 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 0 points. (20 Points)

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

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{i+j}1$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(i+j)$$

$$\sum_{i=1}^{n}\left(ni+\frac{n(n+1)}{2}\right)$$

$$n\left(\frac{n(n+1)}{2}\right)+n\left(\frac{n(n+1)}{2}\right)$$

We know that the n^3 term won't be canceled out, therefore...

Answer: $O(n^3)$

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

Values of i = n, n/2, n/4, ..., 2

The while loop behaves like $\log_2 n$, but within each iteration of the while loop is a summation function. The summation results in n iterations. Therefore, we combine those to get a runtime of $O(n \log_2 n)$.

Answer: $O(n \log_2 n)$

3. **Extra Credit** Consider the following two methods that compute the greatest common divisor of two positive integers. (10 Points)

```
GCD_1(a, b) {
   n = min(a, b); // min(a, b) returns the minimum of a and b
   for (int i = n; i >=1; i--)
      if both (a % i) and (b % i) are zero
         return i;
   }
```

```
GCD_2(a, b) {
    x = max(a, b)
    y = min(a, b)
    while (y != 0) {
       z = x % y;
       x = y;
       y = z;
    }
    return x;
}
```

(a) Use the following pairs of numbers and compute their gcd using the above method. For each pair, report the execution time for each method in a tabular format.

a	b	GCD_1	GCD_2
10234589	98765431	17240542	3167
198491329	217645177	232626458	3583
5915587277	1500450271	1490358000	4917

You can use System.nanoTime() to calculate the execution times.

(b) Discuss why the second algorithm is more efficient than the first. You are not required to provide a proof.

The first algorithm is **less efficient** because it has a big-O runtime of $O(n)$. This is because, in the worst case, the GCD is 1 meaning it would run through the for loop n times. However, the second algorithm has a behavior like $\log n$. Since we know $\log n \in O(n)$, that means the second algorithm is dominated by the first algorithm and is, therefore, more efficient.