Problem Set 0

All parts are due on February 7, 2020 at 6PM. Please write your solutions in the LaTeX and Python templates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct. Compiled solution PDFs should be submitted on Gradescope, and any code should be submitted for automated checking on `alg.mit.edu`.

This assignment is meant to be an evaluation of your **individual** understanding coming into the course and should be completed **without collaboration** or outside help. You **may** ask for logistical help concerning LaTeX formatting and/or code submission.

Problem 0-1. Let $A = \left\{ i + \binom{5}{i} \;\middle|\; i \in \mathbb{Z} \text{ and } 0 \leq i \leq 4 \right\}$ and $B = \{3i \mid i \in \{1, 2, 4, 5\}\}$.

Evaluate: **(a)** $A \cap B$ **(b)** $|A \cup B|$ **(c)** $|A - B|$

Solution: (a) $\{6, 12\}$, (b) 7, (c) 3

Problem 0-2. Let X be the random variable representing the number of heads seen after flipping a fair coin three times. Let Y be the random variable representing the outcome of rolling two fair six-sided dice and multiplying their values. Please compute the following expected values.

Evaluate: **(a)** $\mathrm{E}[X]$ **(b)** $\mathrm{E}[Y]$ **(c)** $\mathrm{E}[X + Y]$

Solution: (a) $3/2 = 1.5$, (b) $49/4 = 12.25$, (c) $55/4 = 13.75$

Problem 0-3. Let $A = 600/6$ and $B = 60 \bmod 42$. Are these statements True or False?

Evaluate: **(a)** $A \equiv B \pmod 2$ **(b)** $A \equiv B \pmod 3$ **(c)** $A \equiv B \pmod 4$

Solution: (a) True, (b) False, (c) False

Problem 0-4. Prove **by induction** that $\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2$, for any integer $n \geq 1$.

Solution: Induct on n. Base Case: for $n = 1$, $1^3 = 1 = \left[\frac{1(1+1)}{2}\right]^2$ as desired. Now assume by induction that $\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2$ is true for $n = k$. We prove the statement is true for $n = k + 1$:

$$\sum_{i=1}^{k+1} i^3 = (k+1)^3 + \sum_{i=1}^{k} i^3 = (k+1)^3 + \left[\frac{k(k+1)}{2}\right]^2 = \frac{(4(k+1) + k^2)(k+1)^2}{4} = \left[\frac{(k+1)(k+2)}{2}\right]^2,$$

reestablishing the inductive hypothesis as desired. □

Problem 0-5. Prove **by induction** that every connected undirected graph $G = (V, E)$ for which $|E| = |V| - 1$ is acyclic.

Solution: Induct on the number of vertices k. Base Case: a graph containing one vertex and zero edges is trivially acyclic. Now assume for induction the claim is true for any connected graph having exactly k vertices and $k-1$ edges, and consider any connected graph G containing exactly $k+1$ vertices and k edges. G is connected so every vertex connects to at least one edge. Since each of the k edges connects to two vertices, the average degree of vertices in G is $2k/(k+1) < 2$, so there exists at least one vertex v with degree 1, connected to exactly one vertex u. Removing v and the edge connecting v to u yields a graph G' on k vertices and $k-1$ edges that is also connected. Vertex v cannot be in any cycle of G since a vertex in a cycle has degree at least 2, so G contains a cycle only if G' contains a cycle. By the inductive hypothesis G' is acyclic, so G is also. □

Problem 0-6. An **increasing subarray** of an integer array is any consecutive sequence of array integers whose values strictly increase. Write Python function `count_long_subarrays(A)` which accepts Python Tuple `A = (`$a_0, a_1, \ldots, a_{n-1}$`)` of $n > 0$ positive integers, and returns the number of longest increasing subarrays of `A`, i.e., the number of increasing subarrays with length at least as large as every other increasing subarray. For example, if `A = (1,3,4,2,7,5,6,9,8)`, your program should return `2` since the maximum length of any increasing subarray of `A` is three and there are two increasing subarrays with that length: specifically, subarrays `(1,3,4)` and `(5,6,9)`. You can download a code template containing some test cases from the website. Submit your code online at `alg.mit.edu`.

Solution:

```
def count_long_subarray(A):
    n = len(A)
    current = 1   # length of longest increasing subarray ending at A[i - 1]
    length = 1    # length of longest increasing subarray in A[:i]
    count = 1     # number of longest increasing subarrays in A[:i]
    for i in range(1, n):
        if A[i - 1] < A[i]:
            current = current + 1
        else:
            current = 1
        if current == length:
            count = count + 1
        elif current > length:
            length = current
            count = 1
    return count
```