CSCI 5454 Spring 2023: Assignment #1
Due Date: Thursday, February 2, 2023.

Topics: Analysis of Algorithms, Euclid's Algorithms and its analysis.

P1. (25 points) Consider the extended Euclid's algorithm that was taught in class.

```
def extended_gcd($a_0$, $b_0$):
    #precond: $a_0 \geq 1, b_0 \geq 0, a_0 \geq b_0$
    (a, b) = ($a_0$, $b_0$) # Initialize
    (s, t) = (1, 0) # Initialize Bezout coefficients for a
    ($\hat{s}$, $\hat{t}$) = (0, 1) # Initialize Bezout coefficients for b
    while ( b > 0 ):
        (q, r) = (a//b, a % b) # compute quotient and reminder
        (a, b) = (b, r) # update a, b
        # update the Bezout coefficients
        (s, t, $\hat{s}$, $\hat{t}$) = ($\hat{s}$, $\hat{t}$, s - q * $\hat{s}$, t - q * $\hat{t}$)
    return (a, s, t)
```

Part A (10 points) Prove the loop invariants:

$$a = sa_0 + tb_0, \text{ and } b = \hat{s}a_0 + \hat{t}b_0$$

Your proof should clearly split into base case and inductive step. Expected solution should follow same format as an example we will post for the class. Unclear proofs will not get any credit.

Part B (5 points) Prove Bézout's Lemma: for all integers $a > 0$ and $b \geq 0$, $\texttt{gcd}(a, b)$ is the smallest positive number that can be written as $sa + tb$ for some integers s, t.

Hint: Break it into two parts. First show that the GCD can be written as $sa + tb$ for some integers s, t. Use the work you did from the previous problem. Next, prove that the GCD divides any positive number of the form $la + ub$. Proof will require 5 lines. Please be precise in your arguments. (Étienne Bézout : `https://mathshistory.st-andrews.ac.uk/Biographies/Bezout/`).

Part C (5 points) Let $n < p$ be natural numbers that are relatively prime (i.e, they have no prime factors in common). Prove that there exists a unique number m (modulo p) such that $n \times m = 1 \mod p$. Proof takes < 10 lines and uses Bézout theorem above. (Note: m is called the modular inverse of n modulo p. We need to perform modular inverse computation as part of RSA public/private key generation).

Part D (5 points) How would you compute the modular inverse of a number n modulo p if n, p are relatively prime? Compute the modular inverse of $n = 13113$ modulo $p = 2133555512$. Hint: use the Bézout coefficients s, t from the extended Euclidean algorithm above.

P2 (25 points). A hedge fund employs many traders for each of whom it compiles a list of returns profits/loss over each trade. For instance, the data corresponding to trader Jane may look like this:

$$[5, 10, -4, 2, 1, 10, 15, -5, -4, -3, 10, 15, -18, 19, 14, 12, 15]$$

This list ℓ of length n represents that Jane earned \$ 5 in her trade number 0, earned \$ 10 in trade number 1, lost \$4 in trade number 2 and so on.

Thus given a list of returns from trades, we wish to support queries where each query inputs two indices (i, j) where $0 \leq i \leq j < n$ and needs to output the sum $\ell[i] + \cdots + \ell[j]$.

Part A (5 points): If you are allowed to store $O(n)$ extra space for Jane, describe how would you use it to support each query in $O(1)$ time? Describe the pre-processing algorithm and the algorithm for handling queries separately. How many steps does your pre-processing need? (expected: 5 lines).

Part B (5 points) Suppose the extra space allowed is reduced to $O(\sqrt{n})$ instead of $O(n)$, describe how you would pre-process and support queries with this new space restriction? How many steps does your pre-processing take? How much time would each query require? (expected : 5 lines).

Part C (10 points) Let us take a small diversion to solve the following problem.

Given a non-empty sorted array of numbers A of length n and a number x, find the index j such that $A[j]$ is the largest number in A such that $A[j] \leq x$. Assume that the inputs satisfy $A[0] \leq x$. We will assume python convention that arrays are indexed from 0 to $n-1$, inclusive.
Example: Suppose $A : [1, 5, 10, 12, 15, 18]$ and $x = 14$, we will return 3 with $A[3] = 12$ as the largest element ≤ 14.

We would like you to modify binary search to find such an index j in time $\Theta(\log(n))$ worst case. Here is the template for the pseudo-code and the loop invariant we would like to establish. Fill in the rest of the outline below so that you are guaranteed a terminating and correct algorithm.

```
def findLargestIndex(A, x):
    assert ( A[0] <= x)
    n = len(A)
    if (x >= A[n-1]):
        return n-1
    # TODO: Initialize two indices l, u
    l = ...
    u = ....
    # Loop Invariants : 0 <= l < u < n and A[l] <= x < A[u]
    while ( ...  ): # fill in the appropriate termination condition
        mid = (l + u) /2 # This will be integer division
        # Compare A[x] to A[mid] and update l, u.
        # Be careful to make sure that  the loop invariants are maintained.
        ...
    return ... # what should we return?
```

Complete the algorithm above and give a brief explanation ($\leq$ 5 lines) of why it works. It will help if you ran it and tested it as well. For the solution, just complete the pseudocode and explain why the loop invariants hold.

Part D (5 points) Returning back to the list of trades ℓ of Jane, describe an algorithm that given two indices $0 \leq i \leq j < n$ will tell us if Jane had any trade k between times i and j that resulted in a loss. In other words, does there exist index $i \leq k \leq j$ such that $\ell[k] < 0$? Your algorithm should return yes/no and for a yes answer it should also return the index k above.

You are allowed $O(n)$ extra space and $O(n)$ time to pre-process the data. Each query should be supported in $O(1)$ time.