1 Problem A: Amalgamated Artichokes

Background

Explaination...

1.1 Mathematical Formulation

Given an input of integers $p, a, b, c, d, andn$, the formula $f(x) = p \cdot (sin(a \cdot x + b) + cos(c \cdot x + d) + 2)$ where $x \in [1, n]$, determine the largest decrease between the integer values x_i, x_j where $i<j$ and $x_i \geq x_j$ and there does not exist another pair x_k, x_l where $k<l$ and $x_k \geq x_l$ but $x_k - x_l > x_i - x_j$.

1.2 Solution

The main functionality of this algorithm is to plug in each point keeping track of the highest seen point, h, the lowest seen point occuring after l, and the largest difference, $d = h - l$. It should be noted that since we are always taking the difference between the two values, we can factor out the $\cdot p$ as well as neglect the $+2$ portions of the formula. Also, to cut down on run time, it works in the java system if you % pi each of the entries before putting them into the sine and cosine functions. For whatever reason the larger the input, the more costly the operation is.

Algorithm 1 Main

```
procedure F(x)
    ab ← (a*x+b) % pi,
    cd ← (c*x+d) % pi;
    return (Math.sin(ab) + Math.cos(cd))
procedure Solve(p, a, b, c, d, n)
    val, h, l ← f(1); diff ← 0
    for x ∈ [2, n] do // if n = 1, do not execute
        val ← f(x)
        if val>h then // higher than current highest
            h, l ← val;
        else if val<l then // lower than current lowest
            l ← val; curDiff ← h - l;
            if curDiff>diff then diff ← curDiff;
    PRINT(p · diff)
```

1.3 Correctness

Proposition 1.
propose

Proof.
Using the fact that □

1.4 Analysis

Proposition 2. *The space complexity of this algorithm is* ***O(1)***

Proof.

This is due to the fact that we will only store the values $p, a, b, c, d, n, anddiff$ as integer variables O(1):

Giving us a space complexity of **O(1)**

□

Proposition 3. *The time complexity of this algorithm is* ***O(N)***

Proof. This is the case because our algorithm goes through the points $1, 2, ..., n$ once and only calculates each value one time.

Giving us a time complexity of **O(N)**

□

1.5 An Example