Question 1)
Constant time:
  10
  n + 10
logarithmic:
    10 log n
Log-linear time:
  10 n log n
  n log n^20
Quadratic:
  n^2 log n + n^3
Cubic time:
    n^3 log n + n^3
    n^3 log n

Question 2)
1 - For each iteration of the outer loop, loop 2 runs for x - y iterations, running linearly with respect to x. However, each iteration of loop 1 shrinks the space that the inner loop needs to iterate over. loop 2 runs in O(x) time per outer loop, with a best case performance of O(1) if y is 1 greater than x.

2 - Loop 1 runs from 1 to n, for a total of n-1 iterations. This is a linear execution with respect to n. The outer loop has a worst case of O(n) and a best case of O(1) if n == 1, as due to the comparison, only a single iteration will occur.

3 - The asymptotic complexity of the two_loops algorithm overall is O(n^2). The outer loop runs in O(n) linear time, leaving the inner loop with a smaller number of iterations each time, but still running linearly. For a given iteration of the outer loop, we see the following:

when x = 1, the inner loop runs n - 1 times
when x = 2, the inner loop runs n - 2 times

The inner loop's two assignment statements both run in O(1) time, as well as incrementing the value of x after the inner loop, and are always considered constant time operations. These are too small to affect the asymptotic complexity of the overall function.

Question 3)
iteration 1: left = 1, right = 64, mid = 1+63/2 = 1+31 = 32, mid_squared = 1024, Left = 1, right = 31

iteration 2: left = 1, right = 31, mid = 1+30/2 = 1+15 = 16, mid_squared = 256, Left = 1, right = 15

iteration 3: left = 1, right = 15, mid = 1+14/2 = 1+7 = 8, mid_squared = 64, Left = 1, right = 7

mid_squared == n, return

The asymptotic complexity should be O(log n) or logarithmic time, as for every iteration, we're shrinking the space between left and right by half, similarly to a binary search through the space. This results in very few iterations needed to find our exit statement.

Question 4)
Lets start by going line by line through the main_algorithm and calculating each line's runtime

To start, line 1 calls `process_a()`, which we know is O(n^2)
Line 2 starts an O(n) loop
Line 3 calls `process_b()`, which we know runs in O(n log n)

Multiplying out the loop, we know the cumulative calls to `process_b()` add up to O(n^2 log n), and then adding on the first call to `process_a()`, we get a total of O(n^2 (n^2 log n)).

We then drop the smallest part, here the `log n`, resulting in O(n^2 * n^2), or O(n^4). This is our final runtime complexity.