COMP S265F Design and Analysis of Algorithms
Lab 3: Fibonacci Numbers, Binary Tree, and Dynamic Programming – Suggested Solution

Question 1.

(a) ***Original***: $(48, 36) \to (12, 36) \to (12, 24) \to (12, 12)$

$\implies g.c.d. = 12$

Improved: $(48, 36) \to (36, 12)$

$\implies g.c.d. = 12$

(b) ***Original***: $(133, 728) \to (133, 595) \to (133, 462) \to (133, 329) \to 133, 196) \to (133, 63)$

$\to (70, 63) \to (7, 63) \to (7, 56) \to (7, 49) \to (7, 42) \to (7, 35)$

$\to (7, 28) \to (7, 21) \to (7, 14) \to (7, 7)$

$\implies g.c.d. = 7$

Improved: $(728, 133) \to (133, 63) \to (63, 7)$

$\implies g.c.d. = 7$

Question 2.

(a) Consider lines 3 to 8 in the while-loop.
Let ℓ be the current length of `num_array`.
Lines 3 and 7 take $O(1)$ time.
Lines 5 to 6 takes $O(1)$ time, so the for-loop in line 4 takes $O(\ell \cdot 1) = O(\ell)$ time.
Line 8 takes $O(\ell)$ time to remove `min_num` from `num_array`.
Thus, the lines 3 to 9 takes $O(1 + \ell + \ell) = O(2\ell + 1) = O(\ell)$ time.

Each iteration of the while-loop will decreases the length of `num_array` by 1.
Therefore, the time complexity of `function` is

$$O(n + (n-1) + \cdots + 1) = O(\frac{n(n+1)}{2}) = O(n^2) \ .$$

(b) The bottleneck is to scan all the items in `num_array` to find the smallest number `min_num` in each iteration of the while-loop, which leads to the quadratic time complexity, i.e., $O(n^2)$ time.

(c) To reduce the time complexity, we can use a $\Theta(n \log n)$-time sorting algorithm, e.g., quick-sort, merge-sort or heap-sort, to sort all the numbers in `num_array` at the beginning.
Then we can simply print all the elements in the sorted array in linear time, i.e., $O(n)$ time.
The time complexity becomes $O(n \log n + n) = O(n \log n)$.

The revised algorithm can be implemented in Python, as follows:

```python
def fuction(num_array):
  num_array.sort()
  for x in num_array:
    print(x)
```