Q1

3 Points

Consider the following code snippet:

```
def demo_function(lissy):
  S = [0] * len(lissy)
  for i in range(len(lissy)-1):
    T = lissy[0 : i + 1]
    S[i] = sum(T) * (i + 2)
  return S
```

Suppose len(lissy) = n. The tightest big-O of this code is:

- ○ $O(\log n)$
- ○ $O(logn * logn)$
- ○ $O(n)$
- ◉ $O(n^2)$

Q2

3 Points

(Multiple answers can be correct)

If $f(n) = 3n \log n + \frac{1}{10}n^5 + 200n^2 + 642n - 700$, then f(n) is:

- ☑ $O(n^6)$
- ☐ $O(nlogn)$
- ☐ $O(n)$
- ☑ $O(2^n)$

Q3

3 Points

Arrange the following functions by growth rate (smallest big O to largest big-O notation). Indicate which functions grow at the same rate (Same Big-O notation).

f1 = 3^n
f2 = $10n^2 - n$
f3 = $n \log n$ + 8n
f4 = 2^{10}
f5 = $n + 100 \log n$
f6 = 2^{n+8}
f7 = $7n - 6$
f8 = $50n \log n$

EXPLANATION

f4: O(1)
f5 = f7: O(N)
f3 = f8: O(NlogN)
f2: O(N^2)
f6: O(2^N)
f1: O(3^N)
So, f4 < f5 = f7 < f3 = f8 < f2 < f6 < f1

Q4

3 Points

Given an n-element sequence S of integers, Algorithm B first chooses $\log n$ elements in S at random, and appends them one-by-one to an initially empty list T. Then, Algorithm B calls Algorithm E on **each** element $T[i]$. Algorithm E runs in $O(i)$ time when it is called on element $T[i]$.

What is the worst-case running time of Algorithm B?

EXPLANATION

Worst-case: O($\log^2 n$)

Q5

8 Points

Implement a function, `bit_flips(a, b)` that determines the number of bits you would need to flip to convert integer a into integer b.

For example:

```
Input: 29 (or 11101), 15 (or 01111)
Output: 2  # because the first and 4th bits from the left are different.
```

```
def bit_flips(a, b):
    """
    :param a: Int -- positive integer
    :param b: Int -- positive integer

    :return: Number of flips needed to convert a into b
    """
    # YOUR CODE
    count = 0

    return count
```

Please note: **no outside package** such as math utility is allowed. `bin()` is also **not allowed**.

- You will get full credit if your program runs in $O(\log n)$ time, in which $n = \max(a, b)$
- You will get 1 bonus point if your program use only bit operators and possibly $+, -, >, <$, or $==$.
- You cannot use the Python math library.

You can copy the code above and paste it into the answer box below, and then work on it:

EXPLANATION

```
def bit_flips(a,b):
    count = 0
    while a > 0 or b > 0:
        if a%2 != b%2:
            count += 1
        a //= 2
        b //= 2
    return count
```

Or

```
def bit_flips(a,b):
    c = a^b
    count = 0
    while c > 0:
        if c&1 == 1:
            count += 1
        c >>= 1
    return count
```