Data Structures - Problem Set

Complexity Analysis

Question 1

Arrange the following functions by growth rate (slowest growth to fastest growth). Indicate which functions grow at the same rate (i.e. f(N) = Θ(g(N)))

f7 = 128	Θ(1)
f4 = $\sqrt{N}$+ 65	Θ($N^{0.5}$)
f1 = 23N	Θ(N)
f5 = 3N	Θ(N)
f8 = (5N)*($\log_2$6N)	Θ(NlogN)
f6 = (N^2/25)*($\log_2$N)	Θ(N^2logN)
f2 = 42N^3	Θ(N^3)
f9 = 2N^3 + 6N	Θ(N^3)
f3 = 6N^8 + 2^N	Θ(2^N)
f10 = $2^{(N+8)}$	Θ(2^N)

Question 2

Give an analysis of the Big-O running time of the following function func1:

```
def func1(n):
        int sum = 0    (1op)
        for i in range (23):    (1op per loop, 23 loops in total)
                for j in range (n):      (1op per loop, n loops in total)
                        sum ++        (2ops per loop per loop)
```

Approximately (3n+1) * 23 +1 = 69n + 24 steps, which is O(n) time complexity.

Question 3

Give an analysis of the Big-O running time of the following function func2:

```
def func2(n):
        int sum = 0    (1op)
        for i in range (n):      (1op per loop, n loops in total)
                for j in range (i, n):   (1op per loop, n-i loops per loop)
                        sum ++        (2ops per loop per loop)
```

Approximately $\sum_{i=0}^{n-1} (3(\mathrm{n}-\mathrm{i})+1) = (3\mathrm{n}+1) \times \mathrm{n} - 3 \times \frac{(n-1)\times n}{2} = 1.5n^2 + 2.5n,$ which is $O(n^2)$ time complexity.

Question 4

At the international conference of epidemiologists, it is tradition that each attendant shakes every other attendant's hand. Assume there are N attendants at the 2019 edition of the conference. *What is the total number of handshakes at the conference?*

$$C_N^2 = \frac{N(N-1)}{2}$$

Question 5

You have an N-story building and plenty of eggs. Suppose that an egg is broken if it is thrown from floor F or higher, and unhurt otherwise. Assume we do not know F.

Q1: Describe a strategy to determine the value of F such that the number of throws is at most log N.

Idea: Binary Search.

Strategy: We first give a throw at the floor mid = N // 2. If the egg breaks, we then give a throw at the floor (0+mid) // 2; Otherwise, we then give a throw at the floor (mid+N) // 2. Continuing this binary process, the worst case is taking $\log_2 N$ number of throws.

Q2: Find a new strategy to reduce the number of throws to at most 2 log F.

Idea: Use 2^i jump to determine the upper bound of F, then use binary search within this upper bound to determine F.

Strategy: Starting from i = 1, i += 1, until $2^{i-1} \le F \le 2^i$. Now 2^{i-1} and 2^i are lower and upper bounds of F respectively. The worst case of this part is taking $\log_2 F$ number of throws. Then we do Binary Search within [2^{i-1}, 2^i]. The worst case of this part is taking $\log_2 F$ number of throws since $2^i - 2^{i-1} = 2^{i-1} \le F$. Hence, the worst case of the whole process is taking $2\log_2 F$ number of throws.

Code:

```python
# Problem 5, Q2

import random

N = 100
F = random.randint(1, 100)

# Find the lower and upper bound
i = 1
while not (2 ** (i - 1) <= F <= 2**i):
    i += 1

bot = 2 ** (i - 1)
top = min(2**i, N)

# Binary Search
while bot <= top:
    mid = (bot + top) // 2
    if F < mid:
        top = mid - 1
    elif F > mid:
        bot = mid + 1
    else:
        break
print(mid == F)  # True
```