Definitions & Equations:

$\forall$ - means "for all"

<u>O(n) – "Upper Bound":</u>

f(n) is O(g(n)) if and only if there exists constants c > 0 and $n_0 \geq 0$ such that:

$$f(n) \leq cg(n), \forall n \geq n_0$$

<u>Ω(n) – "Lower Bound":</u>

f(n) is Ω(g(n)) if and only if there exists constants c > 0 and $n_0 \geq 0$ such that:

$$f(n) \geq cg(n), \forall n \geq n_0$$

<u>Θ(n) – "Tight Bound":</u>

f(n) is Θ(g(n)) if it is both O(g(n)) and Ω(g(n))

OR

f(n) is Θ(g(n)) if there exist constants $c_1, c_2 > 0, n_0 \geq 0$ such that:

$$c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0$$

<u>Arithmetic Summations:</u>

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

<u>Geometric Summations:</u> (for $n \geq 0, a \neq 1$)

$$\sum_{i=0}^{n} a^i = 1 + a + a^2 + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1}$$

T/F: Below you will find several statements. A statement should be marked:

- True (T) if it is always true.
- False (F) if there is at least one counterexample.

T F If f(n) is $\Omega(n^2)$, g(n) is $O(n^2)$, and h(n) = f(n) $*$ g(n), then h(n) is $\Omega(n^4)$.

T F To compare two objects of type *T* in Java, you can use < > and =.

T F A LinkedList implementation of a queue provides an O(1) worst case runtime for enqueue(), dequeue(), and peek().

T F Binary search can be performed on an unsorted array.

T F You should always choose to use the doubling strategy for a growable array and never utilize the +1 strategy.

T F If f(n) is O(n), then f(n) is also $O(n^2)$ and $O(n^3)$.

MCSS: Use the following sequence to answer the questions.

[1, 3, 7, 5, -2, 17, -52, 9, 5, -15, 3, -1]

a. What is the MCSS of this sequence?

b. If the $\Theta(n^2)$ algorithm is used to find the MCSS for this sequence, how many total iterations of the loop will run?

c. If the $\Theta(n)$ algorithm is used to find the MCSS, how many times is the running sum reset? (A.K.A Goes negative)

ADTs & Collections:

Answer the following scenarios with what you deem to be the most appropriate ADT to solve the problem, give a potential implementation for it, and that implementation's runtime to solve the problem, and a brief explanation of your choice.

a. After a couple of weeks of playing Sid Meier's Civilization VI, you decide that you want to see which civilizations you are the most successful with – i.e., which ones you have the most won games with. You don't feel like combing through the logs and manually tallying the wins with each Civilization though, so you decide to write a quick program to help you.

   ADT:
   Implementation:
   Given a civilization's name, runtime to determine wins:
   Brief explanation:

b. Suppose that you have a running wish list of video games you would like to buy, and you can only buy one every two weeks. You decide which ones to buy based on a couple of key factors: whether your friends are playing it, how fun it looks, and the price point. You only ever want to buy the best ranking game every two weeks, but with how many games are constantly coming out, it's getting hard to keep track of all of them…

   ADT:
   Implementation:
   Runtime of deciding what game to buy:
   Brief explanation:

Exact Runtimes and Θ Estimates:

For each of the below snippets of code, give the exact number of times that the sum++ line will be run, as well as a Θ estimate in terms of n.

(Also, feel free to judge me on my VSCode color scheme)

```
int sum = 0;
for (int i = n * n; i > 0; i--) {
    for (int j = 0; j < n - 2; j++) {
        sum++;
    }
}
```

Exact: ______________________________ Big-Theta: Θ(____________)

```
int sum = 0;
for (int i = 1; i <= n; i *= 5) {
    for (int j = 0; j < i; j++) {
        sum++;
    }
}
```

You MAY assume that n is a power of 5.

Exact: ______________________________ Big-Theta: Θ(____________)

```
int sum = 0;
for (int i = 5; i < n; i += 2) {
    for (int j = 0; j < n; j++) {
        sum++;
    }
}
```

You MAY assume that n is odd.

Exact: ______________________________ Big-Theta: Θ(____________)

Big-Oh Proofs:

a. Given that a non-negative function f(n) is O($\frac{1}{3}\mathrm{n}^3$), either prove that $(\mathrm{f(n)})^3$ is O($\frac{1}{4}\mathrm{n}^{10}$) or disprove it using an example.

b. Given that a non-negative function f(n) is O(g(n)), and another non-negative function h(n) is O(f(n)), prove or disprove that h(n) is O(f(n)).