Definitions & Equations:

∀ - means “for all”

O(n) – “Upper Bound”:

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

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

Ω(n) – “Lower Bound”:

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

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

Θ(n) – “Tight Bound”:

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:

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

Arithmetic Summations:

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

Geometric Summations: (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 $\Omega(n^4)$.

Pick $f(n)=n^2$ Pick $g(n)=n$ $h(n)=n^2*n=n^3$

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

Have to use .compareTo(T other)

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

tail pointer insert at end change head look at head

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.

What if memory is limited? What if inserts are very unlikely?

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

True, but not necessarily the most accurate bounds.

MCSS: Use the following sequence to answer the questions.

4 11 16 14 31 0 14 0 3 2

[1, 3, 7, 5, -2, 17, -52, 9, 5, -15, 3, -1] 12 items

a. What is the MCSS of this sequence?

31

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?

i	j	count
0	0 ... 11	12
1	1 ... 11	11
2	2 ... 11	10

$\sum_{i=1}^{12} i = \frac{12(13)}{2} = 6 \cdot 13 = \boxed{78}$

i = 0; i < a.length; i++
j = i; j < a.length; j++

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)

2 times (hits -52, -15)

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.

key → stat

what the stat is

ADT: Map
Implementation: HashMap
Given a civilization's name, runtime to determine wins: O(1)
Brief explanation:
We want to determine wins (value) quickly given a civilization's name (key) making a map ideal. (Knowing the # of wins on their own doesn't make much sense)

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...

we have some kind of collection already...

tells us "how" to prioritize

ADT: Priority Queue
Implementation: Any (LL / array / etc)
Runtime of deciding what game to buy: O(log n)
Brief explanation:
The most frequent use cases of this collection will seemingly be inserts & deletes, but we want to always maintain sorted order and have it run quickly. A Priority Queue provides that for any arbitrary type T.

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++;
    }
}
```

→ runs n^2 times

→ runs $n-2$ times

$n^2(n-2) = n^3 - 2n^2$

Exact: $n^3 - 2n^2$ Big-Theta: Θ(n^3)

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

i	j	count	
1	0	1	5^0
5	0, …, 4	5	5^1
25	0, …, 24	25	5^2
125	0, …, 124	125	5^3
⋮	⋮	⋮	⋮
n	0, …, n-1	n	$5^{\log_5(n)}$

You MAY assume that n is a power of 5. $\sum_{i=0}^{\log_5(n)} 5^i = \frac{5^{\log_5(n)+1} - 1}{5-1} = \frac{(5 \cdot 5^{\log_5(n)}) - 1}{4} = \frac{5n-1}{4}$ (simplifies to n)

Exact: $\frac{5n-1}{4}$ Big-Theta: Θ(n)

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

rewrite this mess →

easier to understand:

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

runs n times

runs $\frac{n-5}{2}$ times

this will be even

$\left(\frac{n-5}{2}\right) n \rightarrow \frac{n^2-5n}{2}$

You MAY assume that n is odd.

Exact: $\frac{n^2-5n}{2}$ Big-Theta: Θ(n^2)

Key: $\exists$ - "there exists" $\equiv$ - "equivalent"
$\forall$ - "for all"

Big-Oh Proofs:

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

Since $f(n)$ is $O(\frac{1}{3}n^3)$, $\exists c$ s.t. $\forall n \geq n_0$, $f(n) \leq cN^3$ by the definition of Big Oh.

It stands then, that when we cube both sides:

$$(f(n))^3 \leq (cN^3)^3 \equiv (f(n))^3 \leq c^3N^9$$

Since c^3 is still some arbitrary constant, we can write this as:

$$(f(n))^3 \leq cN^9 \quad \text{and} \quad cN^9 \leq \frac{1}{4}N^{10}$$

for any N if we pick $c = \frac{1}{4}$. So, $(f(n))^3$ is $O(\frac{1}{4}n^{10})$.

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))~~ O(g(n)).

By definition of Big Oh, we have that:

$$f(n) \leq c_0N \quad \exists c_0 > 0, \forall n \geq n_0$$

for f(n). If we utilize this fact for h(n):

$$h(n) \leq c_1N \quad \exists c_1 > 0, \forall n \geq n_1$$

Since f(n) is O(g(n)), and h(n) is O(f(n)), we can setup an inequality:

$$h(n) \leq c_1N \leq c_0N \quad \exists c_0, c_1 > 0, \forall n \geq n_0, n_1$$

We know that $h(n) \leq c_0N$, which is O(g(n)), so h(n) is O(g(n)).