Disclaimer: This discussion worksheet is fairly long and is not designed to be finished in a single section. Some of these questions of the level that you might see on an exam and are meant to provide extra practice with asymptotic analysis.

THESE ARE TEMPORARY SOLUTIONS WITHOUT ANY REAL EXPLANATIONS PROVIDED FOR THE ASYMPTOTIC QUESTIONS. WE ARE WORKING ON COMPLETING A MORE FORMAL SET OF SOLUTIONS TO HELP PRACTICE/STUDY FOR FUTURE EXAMS.

1 More Running Time

Give the worst case and best case running time in $\Theta(\cdot)$ notation in terms of $M$ and $N$.

(a) Assume that $\text{slam()} \in \Theta(1)$ and returns a boolean.

```java
1 public void comeon() {
2     int j = 0;
3     for (int i = 0; i < N; i += 1) {
4         for (; j < M; j += 1) {
5             if (slam(i, j))
6                 break;
7         }
8     }
9     for (int k = 0; k < 1000 * N; k += 1) {
10        System.out.println("space jam");
11     }
12 }
```

For `comeon()` the best case runtime is $\Theta(N)$ and the worst case is $\Theta(M + N)$.

(b) Exam Practice: Give the worst case and best case running time in $\Theta(\cdot)$ notation in terms of $N$ for `find`.

```java
1 public static boolean find(int tgt, int[] arr) {
2     int N = arr.length;
3     return find(tgt, arr, 0, N);
4 }
5 private static boolean find(int tgt, int[] arr, int lo, int hi) {
6     if (lo == hi || lo + 1 == hi) {
7         return arr[lo] == tgt;
8     }
9     int mid = (lo + hi) / 2;
10    for (int i = 0; i < mid; i += 1) {
11        System.out.println(arr[i]);
12    }
13    return arr[mid] == tgt || find(tgt, arr, lo, mid)
14             || find(tgt, arr, mid, hi);
15 }
```

For `find` the best case runtime is $\Theta(N)$ and the worst case runtime is $\Theta(N^2)$.
2 Recursive Running Time

For the following recursive functions, give the worst case and best case running time in $\Theta(\cdot)$ notation.

(a) Give the running time in terms of $N$.

```java
public void andslam(int N) {
    if (N > 0) {
        for (int i = 0; i < N; i += 1) {
            System.out.println("bigballer.jpg");
        }
        andslam(N / 2);
    }
}
```

For `andslam(N)` the runtime is $\Theta(N)$ in the best and worst case.

(b) Give the running time for `andwelcome(arr, 0, N)` where $N$ is the length of the input array `arr`.

```java
public static void andwelcome(int[] arr, int low, int high) {
    System.out.print("[ ");
    for (int i = low; i < high; i += 1) {
        System.out.print("loyal ");
    }
    System.out.println("]");
    if (high - low > 0) {
        double coin = Math.random();
        if (coin > 0.5) {
            andwelcome(arr, low, low + (high - low) / 2);
        } else {
            andwelcome(arr, low, low + (high - low) / 2);
            andwelcome(arr, low + (high - low) / 2, high);
        }
    }
}
```

For `andwelcome(arr, 0, N)` the runtime is $\Theta(N)$ in the best case and $\Theta(N \log N)$ in the worst case.

(c) Give the running time in terms of $N$.

```java
public int tothe(int N) {
    if (N <= 1) {
        return N;
    }
    return tothe(N - 1) + tothe(N - 1) + tothe(N - 1);
}
```

For `tothe(N)` the runtime is $\Theta(3^N)$ in the best and worst case.

(d) Give the running time in terms of $N$
public static void spacejam(int N) {
    if (N == 1) {
        return;
    }
    for (int i = 0; i < N; i += 1) {
        spacejam(N-1);
    }
}

For to the (N) the runtime is \( \Theta(N!) \) in the best and worst case.

3 Hey you watchu gon do?

For each example below, there are two algorithms solving the same problem. Given the asymptotic runtimes for each, is one of the algorithms guaranteed to be faster? If so, which? And if neither is always faster, explain why. Assume the algorithms have very large input (so \( N \) is very large).

(a) Algorithm 1: \( \Theta(N) \), Algorithm 2: \( \Theta(N^2) \)
(b) Algorithm 1: \( \Omega(N) \), Algorithm 2: \( \Omega(N^2) \)
(c) Algorithm 1: \( O(N) \), Algorithm 2: \( O(N^2) \)
(d) Algorithm 1: \( \Theta(N^2) \), Algorithm 2: \( O(\log N) \)
(e) Algorithm 1: \( O(N \log N) \), Algorithm 2: \( \Omega(N \log N) \)

(a) Algorithm 1: \( \Theta(N) \) - straight forward, \( \Theta \) gives tightest bounds
(b) Neither, something in \( \Omega(N) \) could also be in \( \Omega(N^2) \)
(c) Neither, something in \( O(N^2) \) could also be in \( O(1) \)
(d) Algorithm 2: \( O(\log N) \) - Algorithm 2 cannot run SLOWER than \( O(\log N) \) while Algorithm 1 is constrained on best and worst case by \( \Theta(N^2) \).
(e) Neither, Algorithm 1 CAN be faster, but is not guaranteed - it is guaranteed to be "as fast as or faster" than Algorithm 2.

Why did we need to assume that \( N \) was large? Asymptotic bounds often only make sense as \( N \) gets large, because constant factors may result in a function with a smaller order of growth growing faster than a faster one. For example, take the functions 1000\( n \) and \( n^2 \). \( n^2 \) is asymptotically larger than 1000\( n \), but for small \( n \), it will seem that 1000\( n \) is larger than \( n^2 \).
4 Big Ballin’ Bounds

1. Prove the following bounds by finding some constant \( M > 0 \) and input \( N > 0 \) for \( M \in \mathbb{R}, N \in \mathbb{N} \) such that \( f \) and \( g \) satisfy the relationship.

   (a) \( f \in O(g) \) for \( f = 2n, g = n^2 \)
   
   (b) \( f \in \Omega(g) \) for \( f = 0.1n, g = 40 \)

   (c) \( f \in \Theta(g) \) for \( f = \log(n), g = \log(n^a) \), for \( a > 0 \).

   (a) Let \( M = 2, N = 1 \). Then we see for all \( n \geq N \), \( |f(n)| \leq |Mg(n)| \).
   
   (b) Let \( M = \frac{1}{400}, N = 1 \). Then we see for all \( n \geq N \), \( f(n) \geq |Mg(n)| \). Alternatively, \( M = 1 \) and \( N = 400 \) also satisfies the bound.

   (c) For \( M = \frac{1}{a}, N = 2 \), we have \( |\log(n)| = \left| \frac{1}{a} \times \log(n^a) \right| \). It should be noted that this is an equality, which proves that these functions are asymptotically the same.

2. Answer the following claims with true or false. If false, provide a counterexample.

   (a) If \( f(n) \in O(g(n)) \), then \( 500f(n) \in O(g(n)) \).

   (b) If \( f(n) \in \Theta(g(n)) \), then \( 2f(n) \in \Theta(2g(n)) \).

   (a) True.

   (b) False, consider \( f(n) = n, g(n) = 2n \). Then \( 2f(n) = 2^n \), but \( 2g(n) = 2^{2n} = (2^2)^n = 4^n \).