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.

1 More Running Time

Give an asymptotic bound on 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
public void comeon(int M, int N) {
    int j = 0;
    for (int i = 0; i < N; i += 1) {
        for (; j < M; j += 1) {
            if (slam(i, j))
                break;
        }
    }
    for (int k = 0; k < 1000 * N; k += 1) {
        System.out.println("space jam");
    }
}
```

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

```java
public static boolean find(int tgt, int[] arr) {
    int N = arr.length;
    return find(tgt, arr, 0, N);
}
private static boolean find(int tgt, int[] arr, int lo, int hi) {
    if (lo == hi || lo + 1 == hi) {
        return arr[lo] == tgt;
    }
    int mid = (lo + hi) / 2;
    for (int i = 0; i < mid; i += 1) {
        System.out.println(arr[i]);
    }
    return arr[mid] == tgt || find(tgt, arr, lo, mid)
        || find(tgt, arr, mid, hi);
}
2 Recursive Running Time

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

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

```
1 public void andslam(int N) {
2     if (N > 0) {
3         for (int i = 0; i < N; i += 1) {
4             System.out.println("bigballer.jpg");
5         }
6         andslam(N / 2);
7     }
8 }
```

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

```
1 public static void andwelcome(int[] arr, int low, int high) {
2     System.out.print("[ ");
3     for (int i = low; i < high; i += 1) {
4         System.out.print("loyal ");
5     }
6     System.out.println("]");
7     if (high - low > 1) {
8         double coin = Math.random();
9         if (coin > 0.5) {
10             andwelcome(arr, low, low + (high - low) / 2);
11         } else {
12             andwelcome(arr, low, low + (high - low) / 2);
13             andwelcome(arr, low + (high - low) / 2, high);
14         }
15     }
16 }
```
(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);
}
```

(d) Exam Practice: Give the running time in terms of \( N \)

```java
public static void spacejam(int N) {
    if (N == 1) {
        return;
    }
    for (int i = 0; i < N; i += 1) {
        spacejam(N-1);
    }
}
```
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: $\Omega(N\log N)$

Why did we need to assume that $N$ was large?