1 Warm-up

Determine whether \( f(n) \) is in \( O(g(n)) \), \( \Omega(g(n)) \) or \( \Theta(g(n)) \)

\[
\begin{align*}
  f(n) &= 100 & g(n) &= 1 & f(n) &\in \Theta(g(n)) \\
  f(n) &= n^2 + n & g(n) &= 0.1n^2 & f(n) &\in \Theta(g(n)) \\
  f(n) &= 2^n & g(n) &= 2^{2n} + 100 & f(n) &\in O(g(n)) \\
  f(n) &= n^{100} & g(n) &= 2^n + n \log n & f(n) &\in O(g(n)) \\
  f(n) &= 3\log n & g(n) &= n^2 + n + \log n & f(n) &\in O(g(n)) \\
  f(n) &= n\log n & g(n) &= (\log n)^2 & f(n) &\in \Omega(g(n))
\end{align*}
\]

2 Analyzing Runtime

Give the worst case runtime in \( \Theta(\cdot) \) notation. Extra: Give the best case runtime in \( \Theta(\cdot) \).

A. Give the runtime in terms of \( M \) and \( N \). Assume that \( \text{bump()} \in \Theta(1) \) and returns a boolean.

```java
public void wug() {
    int j = 0;
    for (int i = 0; i < N; i += 1) {
        for (; j < M; j += 1) {
            if (bump(i, j))
                break;
        }
    }
}

public int weirdFib(int N) {
    if (N <= 1) {
        return N;
    }
    return weirdFib(N - 1) + weirdFib(N - 1);
}

public static void mystery(int n) {
    if (n == 1) {
        return;
    }
    for (int i = 0; i < n; i += 1) {
        mystery(n-1);
    }
}
```

Worst case is \( \Theta(M + N) \) and best case is \( \Theta(N) \) for \( wug \). \( \Theta(2^n) \) for \( \text{weirdFib} \) and \( \Theta(n!) \) for \( \text{mystery} \) for both cases.
B. Give the runtime in terms of \( N \), where \( N \) is the length of the input array.

```java
public static boolean mystery(int[] arr) {
    arr = mergesort(arr); // Runs in \( \Theta(N \log N) \), where \( N \) is the length arr
    int N = arr.length;
    for (int i = 0; i < N; i += 1) {
        boolean x = false;
        for (int j = 0; j < N; j += 1) {
            if (i != j && arr[i] == arr[j])
                x = true;
        }
        if (!x)
            return false;
    }
    return true;
}
```

\( \Theta(N^2) \)

C. Give the runtime in terms of \( N \), where \( N \) is the length of \( arr \). Assume \( arr \) is a sorted array of unique elements and that we initially call \( \text{mystery2}(arr, 0, arr.length) \).

```java
public static int mystery2(int[] arr, int low, int high) {
    if (high <= low)
        return -1;
    int mid = (low + high) / 2; // (later, see http://goo.gl/gQI0FN )
    if (arr[mid] == mid)
        return mid;
    else if (mid > arr[mid])
        return mystery2(arr, mid + 1, high);
    else
        return mystery2(arr, low, mid);
}
```

\( \Theta(\log N) \)

What are \( \text{mystery()} \) and \( \text{mystery2()} \) doing? Assuming an int can appear in \( arr \) at most twice, can you rewrite \( \text{mystery()} \) with a better runtime?

\( \text{mystery()} \) returns false if there are unique ints in the array, and true if all ints have a duplicate. A \( \Theta(N) \) algorithm is to XOR all ints together and return true if the result is 0.

\( \text{mystery2()} \) looks for an index \( i \) such that \( arr[i] == i \) and returns it, otherwise it returns -1 if no such index exists.