1 Conceptual Check

Order the following big-O runtimes from most to least efficient:

\(O(n \log n), O(1), O(2^n), O(n^2), O(\log n), O(n), O(n!)\)

\(\quad \subset \quad \subset \quad \subset \quad \subset \quad \subset \quad \subset \quad \subset \)

Are the statements in the right column true or false? If false, correct the asymptotic notation (\(\Omega, \Theta, O\)). \(\Omega(\cdot)\) is the opposite of \(O(\cdot)\), i.e. \(f(n) = \Omega(g(n)) \iff g(n) = O(f(n))\).

- \(f(n) = 100\) \(\quad g(n) = 1\) \(\quad f(n) \in \Omega(g(n))\)
- \(f(n) = n^2 + n\) \(\quad g(n) = 0.1n^2\) \(\quad f(n) \in \Theta(g(n))\)
- \(f(n) = 2^n\) \(\quad g(n) = 2^{n+100}\) \(\quad f(n) \in \Theta(g(n))\)
- \(f(n) = n^{100}\) \(\quad g(n) = 2^n + n \log n\) \(\quad f(n) \in O(g(n))\)
- \(f(n) = 3 \log n\) \(\quad g(n) = n^2 + n + \log n\) \(\quad f(n) \in \Omega(g(n))\)
- \(f(n) = \log n\) \(\quad g(n) = (\log n)^2\) \(\quad f(n) \in O(g(n))\)

2 Analyzing Runtime

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

A. Use \(M\) and \(N\) in your result. \texttt{bump()} is a constant time function that returns a boolean.

```java
int j = 0;
for (int i = 0; i < N; i += 1) {
    for (; j < M; j += 1) {
        if (bump(i, j))
            break; // terminates only the inner loop
    }
}
```

B. Use \(N\) in your result.

```java
public static boolean mystery(int[] arr) {
    arr = bilbosort(arr); // creates sorted copy of arr in \(\Theta(N\log N)\) time
    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; // } on same line for vertical space reasons
```
C. Use $N$ in your result, where $N$ is the length of $arr$. Assume $arr$ is a sorted array of unique elements. Say we call 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/gQI0F"
    if (arr[mid] == mid)
        return mid;
    else if (mid > arr[mid])
        return mystery2(arr, mid + 1, high);
    else
        return mystery2(arr, low, mid);
}
```

Amanda’s Additional for Awesome Asymptotic And Algorithmic Allstars with Alliteration:
What are mystery() and mystery2() doing? Assuming an int can appear in $arr$ at most twice, can you rewrite mystery() with a better runtime?

3 Optimizing Algorithms (Extra Problem)

Given an integer $x$ and a sorted array $A[]$ of $N$ distinct integers, design an algorithm to find if there exists indices $i$ and $j$ such that $A[i] + A[j] == x$.

Let’s start with the naive solution.

```java
public static boolean findSum(int[] A, int x) {
    for (int i = 0; i < ___________________; i++) {
        for (int j = 0; j < ___________________; j++) {
            if (___________________) {
                _____________________;
            }
        }
    }
    return false;
}
```

Can we do this faster? Hint: Does order matter here?

```java
public static boolean findSumFaster(int[] A, int x){
}
```

What is the runtime of both these algorithms?