1 Welcome O & Omega, Θ's Not Alone!

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

$O(\log n)$, $O(1)$, $O(n^n)$, $O(n^3)$, $O(n \log n)$, $O(n)$, $O(n!)$, $O(2^n)$, $O(n^2 \log n)$

_____ $\subset$ _____ $\subset$ _____ $\subset$ _____ $\subset$ _____ $\subset$ _____ $\subset$ _____ $\subset$ _____ $\subset$ _____

Are the statements in the right column true or false? If false, correct the asymptotic notation (Ω, Θ, O). Be sure to give the tightest bound. $\Omega(\cdot)$ is the opposite of $O(\cdot)$, i.e. $f(n) = \Omega(g(n)) \iff g(n) = O(f(n))$.

$f(n) = 20501$	$g(n) = 1$	$f(n) \in O(g(n))$	_____
$f(n) = n^2 + n$	$g(n) = 0.000001n^3$	$f(n) \in \Omega(g(n))$	_____
$f(n) = 2^{2n} + 1000$	$g(n) = 4^n + n^{100}$	$f(n) \in O(g(n))$	_____
$f(n) = \log(n^{100})$	$g(n) = n \log n$	$f(n) \in \Theta(g(n))$	_____
$f(n) = n \log n + 3^n + n$	$g(n) = n^2 + n + \log n$	$f(n) \in \Omega(g(n))$	_____
$f(n) = n \log n + n^2$	$g(n) = \log n + n^2$	$f(n) \in \Theta(g(n))$	_____
$f(n) = n \log n$	$g(n) = (\log n)^2$	$f(n) \in O(g(n))$	_____

2 Analyzing Runtime

Give the worst case and best case runtime for each scenario below.

Use M and N in your result. `ping()` is a constant time, $\Theta(1)$, function that returns an int.

```
int j = 0;
for (int i = N; i > 0; i--) {
    for (; j <= M; j++)
        if (ping(i, j) > 64)
            break;
}
```

Use N in your result, where N is the length of `arr`. Assume `mrpoolsort(arr)` is $\Theta(N \log N)$.

```
public static boolean mystery(int[] arr) {
    arr = mrpoolsort(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;
}
```

Achilles Added Additional Amazing Asymptotic And Algorithmic Analysis Achievements: What is `mystery()` doing?

Using an ADT, can you rewrite `mystery()` with a better runtime? What about if we make the assumption an int can appear in `arr` at most twice, then is there an even better way? (Hint: it uses constant memory)

3 Have You Ever Went Fast?

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:

```
1  public static boolean findSum(int[] A, int x) {
2      for (int i = 0; i < ____________________; i++) {
3          for (int j = 0; j < ____________________; j++) {
4              if (____________________) {
5                  ____________________;
6              }
7          }
8      }
9      return false;
10 }
```

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

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









}
```

What is the runtime of both these algorithms?

4 Basic Interview Type Questions (Extra for Experts)

Union: Write the code that returns an array that is the union between two given arrays. The union of two arrays is a list that includes everything that is in both arrays, with no duplicates. Assume the given arrays do not contain duplicates. Ex: Union of 1,2,3,4 and 3,4,5,6 is 1,2,3,4,5,6
Hint: The method should run in $O(M+N)$ time where M and N are the sizes of the two arrays.

```
public static int[] union(int[] A, int[] B) {




}
```

Intersection: Now do the same as above, but find the intersection between both arrays. The intersection of two arrays is the list of all elements that are in both arrays. Again assume that neither array has duplicates. Ex: Intersection of 1,2,3,4 and 3,4,5,6 is 3,4.
Hint: Think about using ADTs other than arrays to make the code more efficient.

```
public static int[] intersection(int[] A, int[] B) {




}
```

What is the runtime, $\Omega(\cdot)$ and $O(\cdot)$, of the intersection algorithm?