Function Growth

For each of the following pairs of functions, graph them below on the same axis. Your graphs do not have to be precise, but they should accurately reflect relative scale and have proper x-intercepts. In the limit as the input of the functions goes to infinity, which of the two functions dominates the other?

(a) \( f(x) = 500, g(x) = \frac{x}{10} \)
(b) \( f(x) = x^4, g(x) = x^2 \)
(c) \( f(x) = \ln x, g(x) = 3^x \)
Big-O Manipulation

Recall that the definition of Big-O is:

\[ f(n) = O(g(n)) \iff \exists c, x_0, \forall x \geq x_0. f(x) \geq cg(x). \]

That is, \( f(n) = O(g(n)) \) if there exists (\( \exists \)) two constants \( c \) and \( x_0 \) such that for all (\( \forall \)) \( x \)s greater than or equal to \( x_0 \), \( f(x) \geq cg(x) \). Or to put it another way, \( f \) dominates \( g \) times a constant \( c \) after some initial value \( x_0 \). For each of the following pairs of functions, determine if \( f(n) = O(g(n)) \) or \( O(g(n)) = f(n) \). Use this definition of Big-O to prove your claim.

(a) \( f(n) = n + 5, g(n) = n^2 \)
(b) \( f(n) = n!, g(n) = 1000 \)
(c) \( f(n) = n^2, g(n) = \log n \)
Loopy Analysis

For each of the following methods, (a) count the exact number of array accesses each method performs as a function of the input array length $n$ and (b) give the tightest big-O bound possible for that function.

```java
public static int sum1(int[] arr) {
    return arr[0] + arr[1];
}
```

```java
public static int sum2(int[] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        sum = sum + arr[i] + arr[i];
    }
    return sum;
}
```

```java
public static int sum3(int[] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr.length; j++) {
            sum = sum + arr[i];
        }
    }
}
```

```java
public static int sum4(int[] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        for (int j = i; j < arr.length; j++) {
            sum = sum + arr[i];
        }
    }
}
```

If you finish up early, try changing the number of array accesses inside the inner for-loops or their bounds. The number of array accesses surely changes, but what about the big-O bound?
(Additional space for work.)