CS2040: Data Structures and Algorithms

Problems for Week 3 Thursday: Asymptotic Analysis

For: 29 Aug 2024, Tutorial 1

Problem 1. Big-O Time Complexity

Big-O time complexity gives us an idea of the growth rate of a function. In other words, for a large input size N, as N increases, in what order of magnitude is the volume of statements executed expected to increase?

Rearrange the following functions in increasing order of their Big-O complexity. Use $\prec$ to indicate that the function on the left is upper-bounded by the function on the right, and $\equiv$ to indicate that two functions have the same big-O time complexity. An example is given below.

Example. For the following functions:

$5n$	$\frac{1}{2}n^3$	n	$3n^2$

The correct arrangement is

$$n \equiv 5n \prec 3n^2 \prec \frac{1}{2}n^3$$

because $n = O(n)$, $5n = O(n)$, $3n^2 = O(n^2)$ and $\frac{1}{2}n^3 = O(n^3)$.

Now, you try! Rearrange the following 16 functions in ascending order using $\prec$ and $\equiv$.

$4n^2$	$\log_3 n$	$20n$	$n^{2.5}$
$n^{0.00000001}$	$\log n!$	n^n	2^n
2^{n+1}	2^{2n}	3^n	$n \log n$
$100n^{\frac{2}{3}}$	$\log[(\log n)^2]$	$n!$	$(n-1)!$

Problem 2. Time Complexity Analysis

Find the **tightest** big-O time complexity of each of the following code fragments.

Problem 2.a. The big-O time complexity of the following code fragment, in terms of n.

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("*");
    }
}
```

Problem 2.b. The big-O time complexity of the following code fragment, in terms of n.

```
int i = 1;
while (i <= n) {
    System.out.println("*");
    i = 2 * i;
}
```

Problem 2.c. The big-O time complexity of the following code fragment, in terms of n.

```
int i = n;
while (i > 0) {
    for (int j = 0; j < n; j++)
        System.out.println("*");
    i = i / 2;
}
```

Problem 2.d. The big-O time complexity of the following code fragment, in terms of n.

```
while (n > 0) {
    for (int j = 0; j < n; j++)
        System.out.println("*");
    n = n / 2;
}
```

Problem 2.e. The big-O time complexity of the following code fragment, in terms of n and m.

```
String x; // String x has length n
String y; // String y has length m
String z = x + y;
System.out.println(z);
```

Problem 2.f. The big-O time complexity of the following function, in terms of n.

```
void foo(int n){
    if (n <= 1)
        return;
    System.out.println("*");
    foo(n/2);
    foo(n/2);
}
```

Problem 2.g. The big-O time complexity of the following function, in terms of n.

```
void foo(int n){
    if (n <= 1)
        return;
    for (int i = 0; i < n; i++) {
        System.out.println("*");
    }
    foo(n/2);
    foo(n/2);
}
```

Problem 2.h. The big-O time complexity of the following function, in terms of n and m.

```
void foo(int n, int m){
    if (n <= 1) {
        for (int i = 0; i < m; i++) {
            System.out.println("*");
        }
        return;
    }
    foo(n/2, m);
    foo(n/2, m);
}
```