**Assignment #2**

Yaokun Wu

**1.**

Because Big-O notation is the establishment of relative order among functions. Given two functions, there are usually points where one function is less than the other. So it make no sense to unbound the N. Thus, to measure the relative rates of growth, the N >= n0 is needed to say that when N >= n0, one function is bounded by the other one.

**2.**

Based on the definition of Big-O notation,

Let T(N) = f1(N), f(N) = N, then f1(N) = 2N <= c\*f(N) where c >= 2, so that T(N) = f1(N) = O(f(N)) = O(N)

Let T(N) = f2(N), f(N) = N, then f2(N) = 3N <= c\*f(N) where c >= 3, so that T(N) = f1(N) = O(f(N)) = O(N)

It does not matter of what the constant is because one can always find a value of c that satisfy the definition of Big-O notation. And the two functions have the same rate of growth.

**3.**

**a).**

f1(5) = 2 \* 5 = 10

f2(5) = 3 \* 5 = 15

f1(10) = 2 \* 10 = 20

f2(10) = 3 \* 10 = 30

Let x1 = a, x2 = 2 \* x1, then,

f1(x2) = f1(2 \* x1) =2 \* 2 \* a = 2 \* f1(a) = 2 \* f1(x1),

f2(x2) = f2(2 \* x1) = f2(2 \* a) = 3 \* 2 \* a = 2 \* f2(x1), both the result double, the rate of growth is linear.

**b).**

f1(5) = 2 \* 5 \* 5 = 50

f2(5) = 3 \* 5 \* 5 = 75

f1(10) = 2 \* 10 \* 10 = 200

f2(10) = 3 \* 10 \* 10 = 300

Let x1 = a, x2 = 2 \* x1, then,

f1(x2) = f1(2 \* x1) = 2 \* 2 \* x1 \* 2 \* x1 = 2^2 \* f1(x1),

f2(x2) = f2(2 \* x1) = 3 \* 2 \* x1 \* 2 \* x1 = 2^2 \* f2(x1), both the result increased by 2^2, the rate of growth is N^2.

**4.**

The Big-O notation can approximate the rate of growth to evaluate the algorithm we created. The runtime of algorithm is affected much more as the input size grows and Big-O notation can capture this so that we can improve our algorithm if possible.

**5.**

n! grows faster.

lim(n->∞) 2^n/n! = 0, thus, n! grows faster.

**6.**

a. 4n^5 + 3n^2 – 2 O(N^5)

b. 5^n - n^2 + 19 O(5^N)

c. (3/5)\*n O(N)

d. 3n \* log(n) + 11 O(NlogN)

e. [n(n+1)/2 + n] / 2 O(N^2)

**7.**

O(N), because the i increases from 0 to N

**8.**

O(N^2), it is a nested loop where O(f(N) \* g(N)) = O(N \* N) = O(N^2)

**9.**

O(N^2), it is a nested loop where O(f(N) \*g(N)) = O(N \* 2N) = O(N^2)

**10.**

O(N), because the worst case is when the for loop is executed.

**11.**

O(logN), because the loop will execute log2(N) + 1 times till it reach 0.

**12.**

O(logN), because the recursive call will execute log2(N) + 1 times till it reach 0.