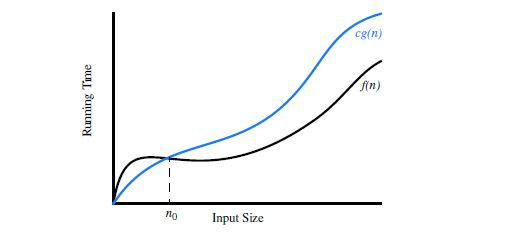
SI: Complexity and Algorithm Analysis

**General Understanding:**

1) In your own words, what is the formal definition of Big O?

2) Based on this definition, can you say that f(n) is O(g(n)) given this graph (assume the c\*g(n) and f(n) never touch again outside the graph)?

g(n)

3) What does the big O tell your about your algorithm

a) How good it is

b) How fast it is

c) How it scales with increasing data

d) How complicated it is to write

5) Given a T(n) of some algorithm, what can you look at the determine the big O? For example, if some algorithm’s T(n) = 4x3 + 4x + 7, what gives away the big O?

**Some Math** (these will help with a few of the problems in the next section) :

1 + 2 + 3 + 4 + ... + N = (N2 + N ) / 2­­

1 + 2 + 4 + 8 + 16 + ... + N/4 + N/2 + N = 2N - 1

**Determining Big O:**

Find the worst case big O for the following algorithms

6)

public int foo(int n) {

int total = 0;

for(int i = 1; i <= n; i++) {

total += i;

}

return total;

}

7)

public int foo(int n) {

int[][] mat = new int[n][n];

int counter = 1;

for(int r = 1; r < n; r\*=2) {

counter++;

}

return counter;

}

8)

public short[] foo(int n) {

short[] result = new short[n];

return result;

}

9)

public int foo(bool[][] mat){

int counter = 0;

//init row and col to some numbers that are in bounds

for(int r = row - 1; r < row + 1; r++) {

for(int c = col - 1; c < col + 1; c++) {

if(mat[r][c])

counter++;

}

}

return counter;

}

10)

public int foo(int n) {

int counter = 0;

for(int i = 1; i < n; i\*=2) {

counter++;

}

return counter;

}

11)

public void foo(int n) {

for(int i = 0; i < n; i++) {

for(int j = 0; j < n; j++) {

SomeClass.*staticMethod1*(); // assume this method is O(N^2)

}

}

}

12)

public void foo(int n) {

for(int i=0; i<n; i++) {

foo(n-1);

}

}

13)

Hint: this is not O(N^2log(n))

public static int foo(int [] data, int tgt) {

int result = 0;

for (int i = 0; i < data.length; i++) {

int tempTgt = tgt \* data[i];

for (int j = 1; j <= data.length; j \*= 2) {

int tot = 0;

for (int k = 0; k < j; k++) {

tot += data[k];

}

if (tot > tempTgt) {

result++;

}

}

}

return result;

}

14)

public int methodC(int[] data, int tgt) {

int r = 0;

final int LIMIT = data.length - 3;

for (int i = 2; i < LIMIT; i++)

if (data[i] < tgt)

for (int k = 1; k < data.length; k \*= 3)

r += data[i] - data[k];

else

for (int k = i - 2; k <= i + 2; k++)

r += data[i] \* data[k];

return r;

}

15)

public int foo(int n) {

int count = 0, total = 0;

for(int i = 0; i < n; i++) {

for(int j = i; j < n; j++) {

count++;

total += count + i \* j;

}

}

return count;

}

**Some more math**

16)

A method is O(N2). It takes 2 seconds for the method to run when N = 2,000. What is the expected time in seconds for the method to run when N = 8,000?

17)

A method is O(Nlog2N). It takes 20 seconds for the method to run when N = 1,000,000. What is the expected time in seconds for the method to run when N = 2,000,000? (remember log21,000,000 is 20 in this class)

18)

The following method takes 2 seconds to complete when n = 250,000. How long do you expect it to complete when n = 750,000? Assume the nextInt method from the Random class is O(1).

public static int[] methodF(int n, Random r)

{

int[] res = new int[n];

for (int i = 0; i < n; i++) {

res[i] = r.nextInt(); //r.nextInt() returns a random integer

}

return res;

}

Answers:

1) T(N) is said to be O( f(N) ) if there exists positive constants can N0 such that T(N) <= cf(N) when N > N0. Or anything close to that

2) Yes

3) C

5) The largest power

6) O(n)

7) O(n2) – Creating an nxn matrix is O(n^2)

8) O(n)

9) O(1)

10) O(log(n))

11) O(n4)

12) O(n!)

13) O(n2)

14) O(nlog(n))

15) O(n2)

15.1) O(n^2)

15.2) O(n)

16) 32

17) 42

18) 6