CS 260P HW2

#2

1. largest input size: n^2 = 3.6\*10^13, so n = 6\*10^6
2. largest input size: n^3 = 3.6\*10^13, so n ~= 33019
3. largest input size: 100n^2 = 3.6\*10^13, so n = 6\*10^5
4. largest input size: nlogn = 3.6\*10^13, so n ~= 1.29\*10^12
5. largest input size: 2^n = 3.6\*10^13, so n ~= 45
6. largest input size: 2^2^n = 3.6\*10^13, so n = ~5

#4

My answer is g1<g3<g4<g5<g2<g7<g6, the reason is:

First, take a look at g6, g7 and g2, we can easily find that 2^n grows faster then n^2 and n^2 grow faster then n. So g6>g7>g2.

For the rest of then, we take log for each of them, and represent logn with z then we get

Log(g5) = (logn)^2=z^2

Log(g3) = logn+loglogn=z+logz

Log(g4)=4/3logn=4/3z

Log(g1)=(logn)^1/2=z^1/2

Log(g2)=n=2^z

So we can conclude that g2>g5>g4>g3>g1, so g1<g3<g4<g5<g2<g7<g6

#5

1. FALSE, for fn = 2, gn = 1, log(fn) = 1, log(gn) = 0. Eventhough fn = O(gn) cause there is a c=4 that fn<gn\*4, log(fn) is not O(log(gn)).
2. FALSE, for fn = 2n, gn = n, 2^fn =2^2n, 2^gn = 2^n. Eventhough fn = O(gn) cause there is a c=4 that fn<gn\*4, log(fn) is not O(log(gn)).
3. TRUE.

#6

(1)For *i*=1, 2,...,*n*

(2) For *j*=*i*+1, *i*+2,...,*n*

(3) Add up array entries *A*[*i*] through *A*[*j*]

(4) Store the result in *B*[*i*, *j*]

(5) Endfor

(6)Endfor

1. Line (1) will be executed n times and line (2) will be executed

(n-1)\*(n-2)…2\*1 = 0.5\*(n\*(n-1)) times. So line (3) here will be executed the same time with line(2) which is 0.5\*(n\*(n-1)) times. But line(3) is self contains a loop which add up A[i] through A[j]. The add instuction will be executed (j-i) times.

When i=1, add instruction being executed for 1+2+…n-1 times;

When i=2, add instruction being executed for 1+2+…n-2 times;

…

when i=n-2, add instruction being executed for 1+2 times;

when i=n-1, add instruction being executed for 1 times;

So we can see that the total time is

(n-1)\*1+(n-2)\*2+…+(n-(n-1))\*(n-1)

= n+2n+(n-1)n-(1^2+2^2+(n-1)^2)

=0.5\*(n^3-n^2)-1/6\*(2n^3-3n^3+n)

=1/6\*(n^3-n).

So we can conclude that total is O(n^3) which means fn = n^3.

1. When n->inf, the total time Tn/fn = 1/6, so therefore the algorithm is also lower bounded by n^3.
2. New algorithm:

Sum = 0;

For i=1, 2, ..., n

Sum = A[i];

For j = i+1, i+2, ..., n

Sum += A[j]

Store the current sum in B[i, j]

Endfor

Endfor

Now, the algorithm is bounded by n^2.