**Allison Neyer**

**Algorithms Hw#2**

**Exercises**

0.1. In each of the following situations, indicate whether f = O(g), or f = Ω(g), or both (in which case

f = Θ(g)).

|  |  |  |  |
| --- | --- | --- | --- |
|  | f(n) | g(n) |  |
|  | n-100 | n-200 | **f = Θ(g)**. both upper and lower bound are the same. Function of n. |
|  | n1/2 | n2/3 | **f = O (n2/3)** |
|  | 100n+logn | n +(logn)2 | **f = Θ(g)**  upper = lower bound are the same.  n+(logn)2 => log(a)b = blog(a)=> 2 loga |
|  | nlogn | 10n log (10n) =  10n log 10+ 10n log n | **f = O(g)** since g(n) is the upper bound of f |
|  | log(2n) = log (2) + log (n) | log (3n) = log (3) + log(n) | since log(3) is slightly bigger than log (2n) but you can multiply function g by a constant where it is both the upper and lower bound, **f = Θ(g)** |
|  | 10 log n | log(n2) = 2 log (n) | **f = Θ(g)** because 30 \* (2 log (n)) > 10 log n > 2 log (n). |
|  | n1.01 = n × n.01 | nlog2n = n × log(n) × log(n) | **f = O (n.01)**  because log(n) x log(n) > n.01 |
|  | = | n(logn)2 | **f = O ((log(n))2)**  because (log(n))2 canbe multiplied by a constant where (log(n))2 **>** n/log(n) |
|  | n.01 | (logn)10 | **f = Ω(g)** because n.01 is a polynomial which dominates (logn)10 |
|  |  | n/logn | **f = Ω(g)** because f > g ultimately. That is the exponential dominates the polynomial. |
|  |  | (logn)3 | f **= Ω(g)** because f > g since polynomials dominate log |
|  |  |  | since exponential g function dominates the polynomial f function (meaning g > f) then g is the upper bound of f so **f = O(g)**. |
|  | n2n | 3n | **f = O(g)** because f< g since g is the bigger exponential. |
|  | 2n | 2n+1 = 21 \*2n => 2n | **f = Θ(g)**  because it’sbasically the same function and can multiply by a constant so the function g is both the upper and lower bound of f. |
|  | n! | 2n | The exponential 2n is the upper bound for the f function. So **f = O(g)** because f< g. |
|  |  |  | So **f = O(g)** because f< g. Since g is the exponential function of an exponential function and f is an exponential of a logarithmic function so since exponential functions are greater than logarithmic f <g. |
|  |  |  | **f = O(g)** because f< g since g is the bigger polynomial. |

f (n) g(n)

(a) n − 100 n – 200

f(n)=g(n)

(b) n1/2 n2/3

(c) 100n + log n n + (log n)2

(d) n log n 10n log 10n

(e) log 2n log 3n (f) 10 log n log(n2 ) (g) n1.01 n log2 n (h) n2/ log n n(log n)2 (i) n0.1 (log n)10

|  |  |  |
| --- | --- | --- |
| (j) | (log n)log n | n/ log n |
| (k) | √n | (log n)3 |

(l) n1/2 5log2 n

(m) n2n 3n (n) 2n 2n+1 (o) n! 2n

2

(p) (log n)log n 2(log2 n)

n

(q)

i=1

ik nk+1

0.2. Show that, if c is a positive real number, then g(n) = 1 + c + c2 + · · · + cn is:

1. Θ(1) if c < 1.

.4a)

p.57 of text

.4b)

Show that *O*(log*n*) matrix multiplications suffice for computing *Xn*. (Hint:Think about computing *X*8.)

*X*=(0111)

<http://math.stackexchange.com/questions/209062/prove-a-bound-on-matrix-multiplication>

I think you are expected to compute the power by using the [Binary Method for Exponentiation.](http://en.wikipedia.org/wiki/Exponentiation_by_squaring). The idea is of great usefulness in computation, so there are many variant executions. We give one which is simple to describe, but somewhat inefficient, because of the storage requirements. However, it is not hard to modify things so that little storage is required.

First calculate the binary expansion of the exponent *n*. So we find bits *a*0,*a*1,*a*2,…,*ak* such that

*n*=*a*020+*a*121+*a*222+⋯+*ak*2*k*.(1)

Calculate *X*0=*I*, *X*1, *X*2, *X*4, and so on up to *X*2*k* by repeated squaring. This works because *X*2*j*+1=(*X*2*j*)2.

Finally, find the product of all the *X*2*i* with *ai*=1. This product is *Xn*. That follows from the fact that *Xs*+*t*=*XsXt*.

The *k* of Formula (1) is of size about log2*n*, so the repeated squarings use about log2*n* matrix multiplications. The multiplications at the end, for the *ai*=1, take (at most) about log2*n* matrix multiplications.

**Remark:** The powers of the particular matrix mentioned in the problem are intimately connected with the Fibonacci numbers. So in particular the binary method for matrix exponentiation is useful for calculating *Fn* for largish *n*. Because of the rapid growth of the Fibonacci numbers, it is best to use exact integer arithmetic.

1.2)

For any arbitrary integer n we can find a p1 such that . If we take log of base 2 on both sides, then we get Similarly, we can find an integer n such that So p1 ≥ the maximum number of digits needed for binary representation. So if p1 is the smallest integer where , then p1 is equal to the maximum digits needed for n’s binary representation. Just like 99 < 103 for any 3 digit number so 3 is max digits needed for representation of 3 digit number. If you take the ratio of p1 to p2, then

= . Cancel out the log10(n) to get = 3.3. If you round that up that is 4.

P1 is max digits needed for binary. P2 is max digits needed for decimal representation. So the ratio is 3.3 and p1 is at most 4 times p2. Therefore, the binary integer is at most four times as long as the corresponding decimal integer.

1.4)

Log(n!) = logn +log(n-1) +…

Log nn = n logn

Log(n!) > lognn

As <http://stackoverflow.com/questions/2095395/is-logn-%CE%98n-logn> says:

log(n!) = log(1) + log(2) + ... + log(n-1) + log(n)

You can get the upper bound by

log(1) + log(2) + ... + log(n) <= log(n) + log(n) + ... + log(n)

= n\*log(n)

And you can get the lower bound by doing a similar thing after throwing away the first half of the sum:

log(1) + ... + log(n/2) + ... + log(n) >= log(n/2) + ... + log(n)

>= log(n/2) + ... + log(n/2)

= n/2 \* log(n/2)

Using the hint:

n! (= 1\*2\*3\*...\*n) is a product of n numbers each less than or equal to n. Therefore it is less than the product of n numbers all equal to n; i.e., n^n.

Half of the numbers -- i.e. n/2 of them -- in the n! product are greater than or equal to n/2. Therefore their product is greater than the product of n/2 numbers all equal to n/2; i.e. (n/2)^(n/2).

Take logs throughout to establish the result.

<http://stackoverflow.com/questions/2095395/is-logn-%CE%98n-logn>

1.11) yes.

page 6 of <http://cs.gmu.edu/~lifei/teaching/cs483_fall08/assignment3.pdf>