CS170–Spring 2020 — Homework HW01 Solutions

lijingqi , SID None

Collaborators: NONE

1.

pass

2.

pass

3.

pass

4. Asymptotic Complexity Comparisons

(a) answer:

note there are some equation

$f_4(n) = 2^{\log_2 n} = n$

$f_5(n) = \sqrt{n} = n^{\frac{1}{2}}$

so result is :

$f_3, f_7, f_2, f_5, f_4, f_9, f_8, f_6, f_1$

(b) answer:

i.

$$f(n) = \log_3 n = \frac{\lg n}{lg3} \tag{1}$$

$$g(n) = \log_4 n = \frac{\lg n}{lg4} \tag{2}$$

because equaiton (1) and (2) diff only a constant factor so $f(n) = \Theta(g)$

ii.

$$f(n) = n \log n^4 = 4n \log n \tag{3}$$

$$g(n) = n^2 \log n^3 = 3n^2 \log n \tag{4}$$

because equation (4) has the higher degree, so $f(n) = \mathcal{O}(g)$

iii.

$$f(n) = \sqrt{n} = n^{\frac{1}{2}} \tag{5}$$

$$g(n) = (\log n)^3 \tag{6}$$

because any polynomial dominates a product of logs, so $f(n) = \Omega(g)$

iv.

$$f(n) = n + \log n \tag{7}$$

$$g(n) = n + (\log n)^2 \tag{8}$$

both f and g grow as $\Theta(n)$ because the linear term dominates the other, so $f(n) = \Theta(g)$

5. Computing Factorials

(a) Find an $f(N)$ so that $N!$ is $\Theta(f(N))$ bits long

m bit number multiply n bit number will get a (m + n) bit number

upper bound

$1 * 2 * 3 * 4 ... N$

$totalbit = \sum_{i=1}^{N} \lg i$

$totalbit < \sum_{i=1}^{N} \lg N = N \lg N$

lower bound

$N/2 * (N/2 + 1) * (N/2 + 2) * ... N$

because N/2 has least $\lg N - 1$ bits

$\lg N/2 + \lg (N/2 + 1) + \lg (N/2 + 2) .. + \lg N > (N/2)(\lg N - 1)$

so the bit number is $f(n) = N \lg N$, this means that our number has $\Theta(N \lg N)$ bits.

(b) Give a naive algorithm to compute $N!$

```
long factorial(int N) {
    long res = 1;
    for(int i = 1; i <= N; i++) {
        res *= i;
    }
    return res;
}
```

Running time:

because res at most $N \lg N$ bit, i at most $\lg N$ bit, so one iteration unit time is $N(\lg N)^2$. there are n iterations, so running time is $\mathcal{O}(N^2 (\lg N)^2)$.

6. Polynomial Evaluation

(a) Describe a naive algorithm that given $[a_0, ..., a_n]$ and x, computes $p_{(}x)$

```
res eval([a0... an], x) {
    res = 0;
    for(int i = 0; i < N; i++) {
        res += ai*x^i;
    }
    return res;
}
```

It takes i time to compute x^i on the ith iteration. There are n iterations. so $\sum_{i=0}^{N} i = \mathcal{O}(n^2)$.

(b) Horner's method

```
res eval([a0... an], x) {
    res = an;
    for(int i = 1; i <= N; i++) {
        res = res * x + an-i
    }
    return res;
}
```

Proof:

at first iteration

$res_1 = a_{n-1} + a_n x$

at $k-1$ th iteration

$res_{k-1} = a_{n-k+1} + x(a_{a-k+2} + ...x(a_{n-1} + xa_n))$

at k th iteration

$res_k = a_{n-k} + res_{k-1}x$

after nth loop iteration , it will have computed the expression $a_0 + x(a_1 + x(a_2 + x(a_3 + ... + x(a_{n-1} + xa_n)))) = \sum_{i=0}^{N} a_i x^i$

when compute res_k, the res_{k-1} already has been compute, so every iteration does an addition and a multiplication, spending constant cost, so running time is $\mathcal{O}(n)$.

about (a) and (b) are all Polynomial Evaluation algorithm, we can see algorithm is faster than (a).