UNIVERSITY WISCONSIN MADISON, COMPUTER SCIENCE DEPARTMENT

# CS577 Assignment 1

Lei Kang, Liang Wang, Yan Zhai

March 20, 2013

## PROBLEM 1

1.1 The answer is $\Theta(NlogN)$
Proof:

- $f(x) = logx$ is monotonic increase on $[1,\infty]$, so:

$$\int_{n-1}^{n} logx\,dx \le logn \le \int_{n}^{n+1} logx\,dx$$

- using integration by parts formula, we have:

$$\int_{1}^{n} logx\,dx = nlogn + O(nlogn)$$

- $f(n) \in O(nlogn)$, because:

$$f(n) = \sum_{i=1}^{n} log\,i \le \sum_{i=2}^{n} \int_{i}^{i+1} logx\,dx = \int_{2}^{n+1} logx\,dx = O(nlogn)$$

- $f(n) \in \Omega(nlogn)$, because:

$$f(n) = \sum_{i=2}^{n} log\,i \ge \sum_{i=2}^{n} \int_{i-1}^{i} logx\,dx = \int_{1}^{n} logx\,dx = \Omega(nlogn)$$

- combined above points, so $f(n) \in \Theta(nlogn)$

1.2 use Stirling approximation and logarithm property, we have:

$$f_8(n) < f_4(n) < f_3(n) < f_7(n) < f_1(n) < f_5(n) < f_2(n) < f_6(n)$$

## PROBLEM 2

2.(a) We have following definitions:

$$T_1(n) = O(n) + T_1(\frac{3n}{4}) + T_1(\frac{n}{4}) \quad (I)$$
$$T_2(n) = 2T_2(n-1) + O(1) \quad (II)$$
$$T_3(n) = T_3((1-\epsilon)n) + O(n^2) \quad (III)$$
$$T_4(n) = \sqrt{n}T_4(\sqrt{n}) + O(n) \quad (IV)$$

2.(b)

i. Using recursion tree method to expand the expression, we could know that at each level of the tree, it would require $h(n) \in O(n)$ time to process. Although the tree is not a full binary tree, we could infer its size should be between a full binary tree with height $log_4 n$ and $log_{\frac{4}{3}} n$. So we have following inequality:

$$h(n) \times log_4 n \le T_1(n) \le h(n) \times log_{\frac{4}{3}} n$$

The lower bound and upper bound differs only in constant coefficient, thus:

$$T_1(n) \in O(h(n)logn) \in O(nlogn)$$

ii. This simple equation could be recursively expanded as:

$$T_2(n) = \sum_{i=2}^{n} O(2^i) = 2^{(n+1)} - 2 = O(2^n)$$

iii. According to master theorem, since the coefficient of subproblem is 1, we could directly infer that:

$$T_3(n) = O(n^2)$$

iv. Again use the recursion tree to expand the $T_4(n)$, we will get a full tree of height $log_2 log_2 n$, and at each level, we need $\sqrt{n} \times O(\sqrt{n}) = O(n)$ to process. So we could easily calculate:

$$T_4(n) = log_2 log_2 n \times O(n) = O(nlog_2 log_2 n)$$

2.(c) $T_4(n) < T_1(n) < T_3(n) < T_2(n)$

## PROBLEM 3

3.1 when $b = 0$, function returns at first 'if' branch, so it means $a^0 = 1$, which is apparently true.

3.2 this inductive base is not as expressive as its alternative form: for $b \le b^*$, the algorithm works. The proof contains two part:

- When $b+1$ is an odd number, *fastExp*$(a, b+1)$ = $a\times$*fastExp*$(a, b)$. Use the inductive base, we have got *fastExp*$(a, b+1) = a^{b+1}$
- When $b+1$ is an even number, *fastExp*$(a, b+1)$ =*fastExp*$(a, \frac{b+1}{2})^2 = (a^{\frac{b+1}{2}})^2 = a^{b+1}$

Now using the induction, we have concluded *fastExp* works for any $n \in \mathbb{N}$

## PROBLEM 4

4.1 one possible solution is:

$$f(n) = (N-1)!,\ \ n \in [(N-1)!, N!),\ \ N \in \mathbb{N}$$
$$g(n) = \frac{n}{2}$$

It's not hard to see that $g(N!+1) < f(N!+1)$ and $g(3N!) > f(3N!)$ when $N$ is large enough, thus $f(n) \notin O(g(n))$ and $g(n) \notin O(f(n))$.

4.2 it suffices to show that for some pair of functions, such $h$ does not exist.
Let $f_1, f_2$ be $f, g$ we defined above. If there is a $h$ so that

$$f_1(n) \in O(h(n))$$
$$f_2(n) \in O(h(n))$$
$$h(n) \in O(f_i(n)), i \in 1,2$$

assume $i$ is 1, then $f_2(n) \in O(h(n)) \in O(O(f_1(n))) \in O(f_1(n))$. But this contradicts the fact $f(n) \notin O(g(n))$ and $g(n) \notin O(f(n))$.

4.3 for any finite set of such functions:

$$\text{if:} h(n) = \sum_{i=1}^{k} f_i(n),$$
$$\text{then for any valid j } f_j(n) \in O(f_j(n)) \in O(\sum_{i=1}^{k} f_i(n)) \in O(h(n))$$

and for any $h'$ satisfies condition(1), we have:

$$\begin{aligned} f_i(n) \in O(h'(n)) &\rightarrow h(n) \in \sum_{i=1}^{k} O(h'(n)) \\ &\rightarrow h(n) \in O(kh'(n)) \\ &\xrightarrow{k \text{ is finite constant}} h(n) \in O(h'(n)) \end{aligned}$$

so $h(n)$ is what we are looking for.