

CS 341

These notes are meant to be supplementary to lecture slides & the textbook, and so may not contain all covered materials. Here I've chosen content which might not be easy to remember and/or is helpful to look at when doing assignments.

Asymptotic Analysis

- **Order Notations:**

- $f(n) \in O(g(n))$ if $\exists c > 0$ and $n_0 > 0$ such that $0 \leq f(n) \leq cg(n) \forall n \geq n_0$
 - f “grows no faster than” g
- $f(n) \in \Omega(g(n))$ if $\exists c > 0$ and $n_0 > 0$ such that $0 \leq cg(n) \leq f(n) \forall n \geq n_0$
 - f “grows no slower than” g
- $f(n) \in \Theta(g(n))$ if $\exists c_1, c_2 > 0$ and $n_0 > 0$ such that $0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \forall n \geq n_0$
 - f and g have the same complexity
- $f(n) \in o(g(n))$ if $\forall c > 0, \exists n_0 > 0$ such that $0 \leq f(n) < cg(n) \forall n \geq n_0$
 - f has lower complexity than g
- $f(n) \in \omega(g(n))$ if $\forall c > 0, \exists n_0 > 0$ such that $0 \leq cg(n) < f(n) \forall n \geq n_0$
 - f has higher complexity than g
- $f \in O(g)$ and $f \in \Omega(g) \iff f \in \Theta(g)$

- **Limit method:** suppose $L = \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$

- $f \in o(g)$ if $L = 0$
- $f \in \Theta(g)$ if $0 < L < \infty$
- $f \in \omega(g)$ if $L = \infty$

- Useful facts for first-principles proofs:

- $\log n \geq 1 \forall n \geq 2$; i.e. $\log n$ grows faster than 1
- $\log n \leq n \forall n \geq 0$; i.e. $\log n$ grows slower than n

- Useful limit laws:

- $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{\log(f(n))}{\log(g(n))}$
- $\lim_{n \rightarrow \infty} f(n)^k = (\lim_{n \rightarrow \infty} f(n))^k$

- Some math rules:

- Summing a polynomial: $\sum_{i=1}^n i^k \in \Theta(n^{k+1})$

- Summing an exponential (special case of geometric series):

$$\sum_{i=1}^n c^i \in \begin{cases} \Theta(c^{n+1}) & \text{if } c > 1 \\ \Theta(n) & \text{if } c = 1 \\ \Theta(1) & \text{if } c < 1 \end{cases}$$

- $a^{\log_b n} = n^{\log_b a}$ (Useful for recursion trees)

- Geometric series:

$$\sum_{i=0}^{n-1} ar^i = \begin{cases} a \frac{r^n - 1}{r - 1} \in \Theta(r^n) & \text{if } r > 1 \\ na \in \Theta(n) & \text{if } r = 1 \\ a \frac{1 - r^n}{1 - r} \in \Theta(1) & \text{if } r < 1 \end{cases}$$

- $a \geq b + c$ if $a \geq 2 \cdot \max(b, c)$ (useful for asymptotic proofs)

- $\lim_{x \rightarrow c} \frac{f(x)}{g(x)} = \lim_{x \rightarrow c} \frac{f'(x)}{g'(x)}$ if it exists (L'Hopital's Rule)

- $\log(n!) \in \Theta(n \log n)$ (Stirling's Approximation)

- $\sum_{i=1}^n \frac{1}{i} \in \Theta(\log n)$ (Harmonic series)

Divide and Conquer

- **Recursion-tree method:**

- Given the recurrence $T(n) = aT(n/b) + f(n), T(1) = c$:
 - a is the # of recursive calls made (# of subproblems)
 - b is the # by which the input size n is divided in each recursive call
 - $f(n)$ is the runtime of the “work done outside of the recursive calls”
 - c is the constant-time work done in each recursive call in the base case
- Each node of the recursion tree represents the cost of the work done other than making recursive calls
- Each row represents the total cost of work done in all recursive calls at that recursion “level”
- The height of the tree depends on the factor that the input size is divided by; i.e. $\log_b n$
- E.g.: Picture this as a tree where each node (except for leaves) has a children;

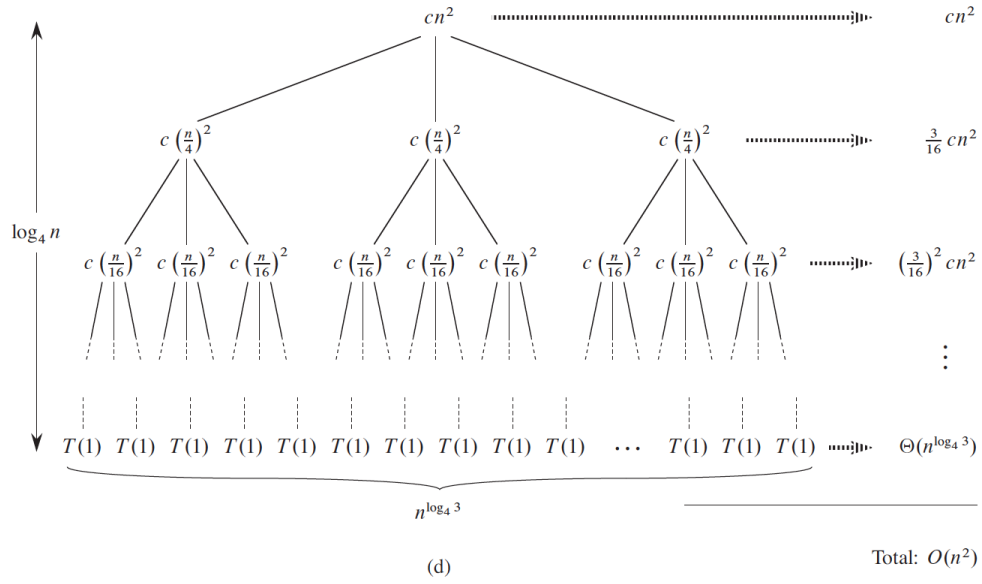
Level 0:	$f(n)$			total = $f(n)$
Level 1:	$f(n/b)$...	$f(n/b)$	total = $af(n/b)$
Level 2:	$f(n/b^2)$...	$f(n/b^2)$	total = $a^2f(n/b^2)$
...				
Level k :	$f(n/b^k)$...	$f(n/b^k)$	total = $a^k f(n/b^k)$
...				
Level $\log_b n$:	c	...	c	total = $ca^{\log_b n} = cn^{\log_b a}$

- Total runtime of recursion tree is (summing every row total):

$$T(n) \in \Theta \left(\sum_{i=0}^{\log_b n - 1} a^i f(n/b^i) \right) + \Theta(n^{\log_b a})$$

- Use geometric series formula to find a simplified value

- Example: $T(n) = 3T(n/4) + n^2$



$$T(n) \in \Theta \left(\sum_{k=0}^{\log_4 n - 1} \left(\frac{3}{16} \right)^k n^2 + n^{\log_4 3} \right) \in \Theta(n^2)$$

- **Master method:**

- Given the recurrence $T(n) = aT(n/b) + f(n)$ where $f(n) \in \Theta(n^d)$:

$$T(n) \in \begin{cases} \Theta(n^{\log_b a}) & \text{if } a > b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^d) & \text{if } a < b^d \end{cases}$$

Greedy Algorithms

- Two methods of proving greedy algorithms:
- **Greedy stays ahead (induction)**
 - Show that S_g is better than S at every step
 - Base case: show $S_g[1] > S[1]$
 - Inductive hypothesis: assume $S_g[k-1] > S[k-1]$; show that $S_g[k] > S[k]$
 - Often by contradiction; i.e. suppose $\exists S^*$ such that $S_g[k] < S^*[k]$, and derive some contradiction
- **Exchange argument (swapping)**
 - Let S_g = greedy solution, S = some arbitrary solution
 - Show that (any) S can be transformed into S_g step-by-step without getting worse at any point
 - i.e. compare the cost of before & after swapping 2 elements in S , show that it doesn't change or improves