CS3230 final reference

Isaac Lai

April 30, 2024

1 Asymptotic facts

$$
\begin{aligned}
e^x &\geq 1 + x \\
a^{\log_b c} &= c^{\log_b a} \\
n! &= \sqrt{2\pi n}\left(\frac{n}{3}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right) \text{ (Stirling's approximation)} \\
\log(n!) &= \Theta(n \log n) \\
\sum_{k=0}^{n} ar^k &= \frac{a(r^{n+1} - 1)}{r - 1} \text{ (Geometric series)} \\
\sum_{k=1}^{n} \frac{1}{k} &= \ln n + O(1) \text{ (Harmonic series)} \\
\lim_{x\to\infty} \frac{f(x)}{g(x)} &= \lim_{x\to\infty} \frac{f'(x)}{g'(x)} \text{ (L'Hopital's Rule)}
\end{aligned}
$$

2 Asymptotic analysis

- $f(n) = O(g(n))$ if $\exists c > 0, n_0 > 0$ such that $\forall n \geq n_0, 0 \leq f(n) \leq cg(n)$
- $f(n) = \Omega(g(n))$ if $\exists c > 0, n_0 > 0$ such that $\forall n \geq n_0, 0 \leq cg(n) \leq f(n)$
- $f(n) = \Theta(g(n))$ if $\exists c_1, c_2 > 0, n_0 > 0$ such that $\forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ i.e. $\Theta(g) = O(g) \cap \Omega(g)$
- $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0 \implies f(n) = o(g(n))$
- $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} < \infty \implies f(n) = O(g(n))$
- $0 < \lim\limits_{n\to\infty} \frac{f(n)}{g(n)} < \infty \implies f(n) = \Theta(g(n))$
- $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} > 0 \implies f(n) = \Omega(g(n))$
- $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \infty \implies f(n) = \omega(g(n))$

3 Recurrences

General form: $T(n) = aT(\frac{n}{b}) + f(n)$

- Telescoping: express in form $\frac{T(n)}{g(n)} = \frac{T(\frac{n}{b})}{g(\frac{n}{b})} + h(n)$
- Recursion tree: draw tree, sum each node (can sum over level first then over height)
- **Master Theorem**: $a \geq 1, b > 1, f$ asymptotically positive
  1. $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$. $T(n) = \Theta(n^{\log_b a})$
  2. $f(n) = \Theta(n^{\log_b a} \log^k n)$ for some $k \geq 0$. $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
  3. $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and satisfies **regularity condition** $af(\frac{n}{b}) \leq cf(n)$ for some $c < 1$. $T(n) = \Theta(f(n))$
- Substitution: guess and check by induction. Induction hypothesis: $c_1 n^k -$(lower order terms)

4 Divide and Conquer

Invariant: condition which is true at the start of every iteration. To show correctness check

- Initialisation: invariant is true at iteration 1
- Maintenance: if invariant is true for iteration n, it remains true for iteration $n + 1$
- Termination: when the algorithm ends, the invariant helps the proof of correctness

Strassen's algorithm: matrix multiplication in $O(n^{\log_2 7})$ (splitting into 7 multiplications instead of 8)

5 Sorting

Lower bound for **comparison-based** sort is $O(n \log n)$ by argument from decision tree

No. of comparisons	Mergesort	Quicksort
Average case	$\Theta(n \log_2 n)$	$\Theta(n \log_2 n)$
Best case	$\Theta(n \log_2 n)$	$\Theta(n \log_2 n)$
Worst case	$\Theta(n \log_2 n)$	$\Theta(n^2)$

6 Randomisation

- Las Vegas algorithms: (1) output always correct (2) running time depends on random bits, with small probability that it may be large (3) expected running time is bounded by given time-bound function
- Monte Carlo algorithms: (1) answer may be incorrect with small probability (2) running time is always bounded by given time-bound function

- Union bound: $P(A \cup B) \leq P(A) + P(B)$
- Linearity of expectation: $E[X + Y] = E[X] + E[Y]$ even if X and Y are not independent random variables

7 Dynamic Programming

- Optimal substructure: optimal solution contains optimal solutions to subproblems
- Overlapping subproblems: recursive solution contains a small number of distinct subproblems repeated many times
- Top-down saves computation of unnecessary subproblems but can suffer from overhead of recursive calls, bottom-up is the opposite

8 Greedy

- Greedy choice: pick the largest/smallest/etc. (some extreme value). Show there is always an optimal solution to the original problem that makes the greedy choice
- Use optimal substructure to show we can combine an optimal solution to the subproblem with the greedy choice to get an optimal solution to the original problem

9 Amortised analysis

- Shows that the average cost per operation is small, even though a single operation within the sequence might be expensive
- Accounting method: charge ith operation an amortised cost $c(i)$. Must ensure the sum of true costs $\sum_{i=1}^{n} t(i) \leq \sum_{i=1}^{n} c(i)$
- Potential method: $\phi(0) = 0$, $\phi(i) \geq 0$ for all i. Amortised cost of ith operation $= t(i) + (\phi(i) - \phi(i-1))$. Heuristic: try to find some quantity that is "decreasing" during the operation

10 Reductions and NP-completeness

- Poly-time: polynomial in terms of the length of the encoding of the problem instance
- Pseudo-polynomial: polynomial in numeric value of the input, but not necessarily in length of the input
- $A \leq_p B$: A is poly-time reducible to B. So if B has a poly-time algorithm, so does A. Conversely, A can be seen as a special case of B, so if A is hard, then B is hard too
- Decision reduces to optimisation (given value of optimal solution, simply check if it is $\leq k$)
- NP-complete: problem must be both in NP and NP-hard

- NP-complete problems: Circuit SAT, CNF-SAT, 3-SAT, MAX-2-SAT, Vertex Cover, Independent Set, Max-Clique, (Directed/Undirected) Hamiltonian Cycle, Travelling Salesperson, Parition, Subset Sum, 0-1 Knapsack

11 Order statistics

- Worst case linear time algorithm to select the rank-i element