1 Function Runtime Analysis

For comparing the asympototic runtime of two functions $f(n)$ and $g(n)$:

- $f(n) = O(g(n))$ if $\lim_{x\to\infty} \frac{f(n)}{g(n)} < \infty$. "Big O"
- $f(n) = \Omega(g(n))$ if $\lim_{x\to\infty} \frac{f(n)}{g(n)} > 0$. "Big Omega"
- $f(n) = o(g(n))$ if $\lim_{x\to\infty} \frac{f(n)}{g(n)} = 0$. "Little O"
- $f(n) = \omega(g(n))$ if $\lim_{x\to\infty} \frac{f(n)}{g(n)} = \infty$. "Little Omega"
- $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. "Theta"

2 Divide-and-Conquer Strategy

1. Break it into subproblems that are themselves smaller instances of the same tyoe of problem.
2. Recursively solve these subproblems.
3. Appropriately combine their answers.

3 Master Theorem

For constants $a > 0$, $b > 1$, $d \geq 0$, the recurrence $T(n) = aT(\frac{n}{b}) + O(n^d)$ solves to:

Crib Sheet

$$T(n) = \begin{array}{ll} O(n^d) & \text{if } \frac{a}{b^d} < 1 \\ O(n^d \log n) & \text{if } \frac{a}{b^d} = 1 \\ O(n^{log_b a} & \text{if } \frac{a}{b^d} > 1 \end{array}$$

Textbook

$$T(n) = \begin{array}{ll} O(n^d) & \text{if } d > log_b a \\ O(n^d \log n) & \text{if } d = log_b a \\ O(n^{log_b a} & \text{if } d < log_b a \end{array}$$

$a =$ branching factor, number of subproblems created
$n/b =$ size of subproblems
$O(n^d)$ time it takes to combine subproblem results

Height of recursion: $log_b n$ levels
Width of tree: $n^{log_b a}$
kth level of the tree is made up of a^k subproblems, each of size n/b^k
Total work at kth level: $a^k \times O(\frac{n}{b^k})^d = O(n^d) \times (\frac{a}{b^d})^k$

Examples: $T(n) = 7T(n/4) + n$. Answer: $\Theta(n^{\log_4(7)})$. — $T(n) = 7T(n/4) + n^2$. Answer: $\Theta(n^2)$ — $T(n) = 64T(n/3) + \sqrt{n}$. Answer: $\Theta(n^{\log_3(64)})$

4 Big-O of Geometric Series

$$\sum_{i=0}^{k} a^i = \begin{cases} O(1) & \text{if } a < 1 \\ O(k) & \text{if } a = 1 \\ O(a^k) & \text{if } a > 1 \end{cases}$$

For $a \neq 1$, $1 + a + a^2 + \ldots + a^n = \sum_{i=0}^{n} a^i = \frac{1-a^{n+1}}{1-a}$

5 Manipulating logs

1. $\log_b n^c = c \log_b n$
2. $\log_b(n \cdot m) = \log_b n + log_b m$
3. $\log_x y = \frac{\log_z y}{\log_z x}$ for any Z

4. $2^{\log_3 n} = (3^{\log_3 2})^{\log_3 n} = (3^{\log_3 n})^{\log_3 2} = n^{\log_3 2}$
5. $\frac{d}{dx} \ln x = \frac{1}{x}$
6. $\frac{d}{dx} \log_a x = \frac{1}{x \ln a}$

6 Proofs

1. Proof by Induction
   Want to prove that some fact is true for all integers n
2. Proof by Contradiction
   If you want to prove that something is T, 1. Assume it is F, 2. Find a contradiction implied by assuming it is F
3. Contraposition
   If you want to prove $A \rightarrow B$, you can prove $\neg B \rightarrow \neg A$
4. Direct Proof
   Logically straight forward proof
5. Counter example
   In order to show that a blanket statement is false, you need to find only 1 counterexample that breaks the statement

Induction
Claim: $f(n)$ returns $n!$
Base Case: when $n = 1$ $f(n)$ returns 1
Inductive step: Assume claim is true for all values $< n$
When $n > 1$ $f(n)$ returns $n * f(n-1)$ by the inductive hypothesis
$f(n-1) = (n-1)!$ therefore $n * f(n-1)$
$= n * (n-1)! = n * (n-1)(n-2).... * 1 = n!$

Contradiction
Claim: there are infinitely many prime numbers
For sake of contradiction, assume there are a finite number of primes
Consider $(p_1 * p_2 ... * p_n + 1) = k$
1) k isn't divisible by p_i, $k \bmod p_i = 1$
Contradiction: 1) k is prime or 2) K is divisible by things not on our list $1....n$

7 Greedy Algorithms

1. Coins - counter example: den = 1, 10, 25. Greedy: 25, 1, 1, 1, 1, 1 Optimal: 10, 10, 10
2. Knapsack - Greedy Algo: (1) Calculate value/weight for each item. (2) Sort from highest to lowest ratio. (3) For each item in list, add it to bag if it fits. Counter Example: Bag size = 10. Item(weight, cost). Item0(6, 7000), Item1(5, 4000). Density: 7000/6 = 1166. 4000/5 = 800. Greedy: (6, 7000) Optimal: 2 times (5, 4000).
3. Activity Scheduling: - choose earliest ending time

```
function Schedule(Array Activities[A_1, ... A_n])
    Sort Activities by increasing end times
    curend = 0
    totevents = 0
    for i ∈ [1, ..., n] do
        if A_i.start ≥ curend then          ▷ If TRUE, we are "choosing" this activity
            curend = A.end
            totevents++
    return TotEvents
```

   Base case: say there was only one activity. Then our algorithm would choose that one activity on the one and only pass through the for loop. This is clearly optimal. Inductive step: say that we have $n > 1$ activities total,and that this algorithm is optimal for up to $[1, \ldots, n-1]$ activities.
   First, we argue that A_1 MUST be in an optimum solution. Consider an optimum solution that does not include A_1, but whose first element is some A_k. Since A_1 ends before A_k, we could replace A_k with A_1 without interfering with any of the activies that come after A_k. Therefore, we may as well choose A_1 as the first element!