The cheatsheet will consist of the (1) defintions and theorems, (2) process, (3) canonical example with mark distribution.

1 Asymptotics

1.1 Big-O (Upper bound)

Definition: $f(n) = O(g(n))$ iff $\exists$ positive constants c and n_0 s.t. $0 \leq f(n) \leq cg(n)\ \forall\ n \geq n_0$

1.2 Big-Omega (Lower bound)

Definition: $f(n) = \Omega(g(n))$ iff $\exists$ positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)\ \forall\ n \geq n_0$

1.3 Big-Theta (Tight bound)

Definition: $f(n) = \Theta(g(n))$ iff $\exists$ positive constants $c_1,\ c_2,\ n_0$ s.t. $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall\ n \geq n_0$ $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

1.4 Small-o (Strictly slower)

Definition: $f(n) = o(g(n))$ iff $\forall c > 0, \exists n_0 > 0$ s.t. $0 \leq f(n) < cg(n)$ for all $n \geq n_0$.

1.5 Small-Omega (Strictly Faster)

Definition: $f(n) = \omega(g(n))$ iff $\forall c > 0, \exists n_0 > 0$ s.t. $0 \leq cg(n) < f(n)$ for all $n \geq n_0$.

1.6 Comparing function properties

Definition:
Transitivity:
- $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$ imply $f(n) = \Theta(h(n))$
- $f(n) = O(g(n))$ and $g(n) = O(h(n))$ imply $f(n) = O(h(n))$
- $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n))$ imply $f(n) = \Omega(h(n))$

Symmetry:
- $f(n) = \Theta(g(n))$ iff $g(n) = \Theta(f(n))$.

Transpose symmetry:
- $f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$

Different functions:
- $n^a = O(n^b)$, iff $a \leq b$.
- $\log_a(n) = O(\log_b(n))$, $\forall\ a, b$.
- $c^n = O(d^n)$, iff $c \leq d$.
- If $f(n) = O(f'(n))$ and $g(n) = O(g'(n))$, then:
  1. $f(n) \cdot g(n) = O(f'(n) \cdot g'(n))$.
  2. $f(n) + g(n) = O(\max\{f'(n), g'(n)\})$.
     - ' is not a derivative, just another function.

1.7 Limit method

Definition: Find the asymptotic relationship between two functions for which you might not have any intuition about.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = o(g(n)) \tag{1}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f(n) = \omega(g(n)) \tag{2}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \text{ i.e. is anything finite } \Leftrightarrow f(n) = O(g(n)) \tag{3}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0 \text{ i.e. non-zero } \Leftrightarrow f(n) = \Omega(g(n)) \tag{4}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = C \text{ s.t. } 0 < C < \infty \Leftrightarrow f(n) = \Theta(g(n)) \tag{5}$$

1.8 Polynomially-bounded

Definition:
- **Polylogarithmically bounded:** $f(n) = O((\lg n)^k) \quad \exists k > 0$
- **Polynomially bounded:** $f(n) = O(n^k) \quad \exists k > 0$
- **Exponentially bounded:** $f(n) = O(k^n) \quad \exists k > 0$

Theorem: All logarithmically bounded functions are polynomially bounded, i.e. $(\lg n)^a = O(n^b) \quad \forall a, b > 0$

Theorem: All polynomially bounded functions are exponentially bounded, i.e. $f(n) = O(n^a) \Rightarrow f(n) = O(b^n) \quad \forall a > 0$ and $\forall b > 1$

1.9 Logarithm method

1.9.1 Limits of logs are logs of limits

Definition: $\lim_{x\to a} (\log_b f(x)) = \log_b \left(\lim_{x\to a} f(x) \right)$

Process: Suppose we want to compute $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L$

1. **Take log of limit:**
$$\lg \left(\lim_{n\to\infty} \frac{f(n)}{g(n)} \right) = \lg L$$
2. **Change to limit of log and compute it:**
$$\lim_{n\to\infty} \left(\lg \frac{f(n)}{g(n)} \right) = \lg L$$
3. **Revert log by taking exponential with base 2:**
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 2^{\lg L} = L$$

Process:

2 Logarithms

3 Induction, Contradiction, & Combinatorial Arguments

3.1 Direct proof

Process:
1. Start with the givens
2. Mathematically manipulate the givens and/or reason about the givens to arrive at the conclusion.

3.2 Weak Induction

Process: Given predicate $P(n)$
1. **Basis Step:** Prove $P(n_0)$ for some value n_0.
2. **Hypothesis:** Assume true for $P(n)$ for a $n = k$.
3. **Inductive step:** Use the hypothesis to show its true for $P(n = k) \implies P(n + 1 = k + 1)$.

Therefore, $\forall n \geq c$, $P(n)$.

3.3 Strong Induction

Process: Given predicate $P(n)$
1. **Basis:** Show $P(n_0), P(n_1), \ldots$ are true.
2. **Hypothesis:** Assume $P(k)$ is true, $\forall k \leq n$.
3. **Step:** Show $P(n_0) \wedge \cdots \wedge P(k) \wedge \cdots \wedge P(n) \Rightarrow P(n + 1)$.

3.4 Contradiction

Process: Given predicate $P(n)$ either true or false.
1. Assume toward a contradiction $\neg P(n)$.
2. Make some argument by working with the expression $\neg P(n)$ to get to a contradiction.
3. Arrive at a contradiction
4. If this resulted in a contradiction then $P(n)$ is true.

4 Recurrences and the Master Theorem

5 Heaps & Heapsort

6 Quicksort

7 Sorting and Searching in Linear Time

8 BSTs & RBTs

9 Hashing

10 Dynamic programming

Process:

1. Visualize example.
2. **Optimal substructure:** Characterize the structure of an optimal solution)
3. **Recursive formula:** Find a relationship among sub-problems (i.e. defines the values of an optimal solution recursively in terms of the optimal solution to sub-problems)
   (a) Base case(s)
   (b) Recursive formula
4. Compute the value of an optimal solution (bottom-up solving sub-problems in order or top-down solving problem recursively)
5. **Time complexity:** $O(n^{\# \text{ subproblems per choice}})O(\# \text{ choices})$

10.1 How to prove optimal substructure?

Process:

Example:

11 Greedy Algorithms

Process:

1.

Example: