COMP3121 Lecture Notes Summary

Module 1: Foundations

- **Course Admin**: Paul Hunter (K17-217D), `cs3121@unsw.edu.au` (Song, Liam, James)
- **Learning Outcomes**:
  - Identify algorithm features
  - Solve problems using data structures and algorithms
  - Communicate algorithmic ideas
  - Evaluate algorithm efficiency and correctness
  - Use LaTeX for technical documents

Time Complexity

Key Concepts

- **Best/Worst/Average Case Performance**:
  - Worst-case is most common (robust against bad inputs)
  - Best-case is rarely discussed
  - Average-case requires probabilistic methods (e.g., quicksort, hash tables)
- **Asymptotic Analysis**:
  - Focuses on long-term growth rates (ignores small inputs)
  - Dominant term determines scalability

Notations

- **Big-Oh (O)**: Upper bound ($\leq$)
  - E.g., $2n + 7 = O(n^2)$
- **Big-Omega (Ω)**: Lower bound ($\geq$)
  - E.g., finding max in an unsorted array is $\Omega(n)$
- **Big-Theta (Θ)**: Tight bound ($=$)
  - E.g., mergesort is $\Theta(n \log n)$

Properties

- **Sum**: $f_1 + f_2 = O(\max(g_1, g_2))$
- **Product**: $f_1 \cdot f_2 = O(g_1 \cdot g_2)$

Data Structures

Binary Heaps

- **Max/Min Heap**: Complete binary tree with parent $\geq$ ($\leq$) children
- **Operations**:
  - Build heap: $O(n)$
  - Insert/Delete max: $O(\log n)$
  - Find max: $O(1)$

Binary Search Trees (BST)

- **Operations**: Search/Insert/Delete in $O(h)$ time
- **Self-Balancing BSTs**: Guarantee $h = O(\log n)$ (e.g., AVL, Red-Black trees)

Hash Tables

- **Expected Time**: $O(1)$ for search/insert/delete
- **Worst Case**: $O(n)$ (due to collisions)
- **Collision Handling**: Separate chaining (linked lists at each index)

Proof Techniques

Propositional Logic

- Operations: $\neg P$, $P \wedge Q$, $P \vee Q$, $P \rightarrow Q$
- Quantifiers: $\forall$ (for all), $\exists$ (exists)

Induction

- Base case + inductive step to prove a sequence of propositions
- **Strong Induction**: First k propositions imply the $(k+1)$th

Contradiction

- Assume $\neg P$, derive a contradiction to prove P
- **Example**: Stable matching in Gale-Shapley algorithm

Algorithm Proofs

- **Correctness**: Always produces the right answer
- **Efficiency**: Runs in claimed time complexity

Stable Matching Problem

Definitions

- **Perfect Matching**: All engineers paired
- **Stable Matching**: No unmatched pair prefers each other over current partners

Gale-Shapley Algorithm

- **Process**:
  - Frontend engineers pitch to backend engineers in preference order
  - Backend engineers accept or reject based on preferences
- **Claims**:
  - Terminates in $\leq n^2$ rounds
  - Produces a perfect matching
  - Matching is stable (proof by contradiction)

Puzzle

- **Problem**: Circular highway with n petrol stations; total fuel = one lap
- **Goal**: Prove there exists a starting station to complete the lap without running out of fuel

Key Takeaways

- Asymptotic analysis is crucial for comparing algorithms
- Data structures have trade-offs (e.g., BST vs. hash tables)
- Proofs ensure correctness and efficiency of algorithms
- Gale-Shapley guarantees stable matchings efficiently