- Identify the **worst case** and **best case** inputs of functions
- Compare the **function families** that characterize different functions
- Calculate a specific function or algorithm's efficiency using **Big-O notation**

- Identify core parts of **trees**, including **nodes**, **children**, the **root**, and **leaves**
- Use **binary trees** implemented with dictionaries when reading and writing code

- Identify core parts of **graphs**, including **nodes**, **edges**, **neighbors**, **weights**, and **directions**.
- Use **graphs** implemented as dictionaries when reading and writing simple algorithms in code

- Identify whether a tree is a **binary search tree**
- Search for values in BSTs using **binary search**
- Analyze the **efficiency** of binary search on a **balanced** vs. **unbalanced** BST
- Search for paths in **graphs** using **breadth-first search** and **depth-first search**
- Analyze the **efficiency** of BFS and DFS on a graph

- Identify **brute force approaches** to common problems that run in **O(n!)** or **O(2n)**, including solutions to **Travelling Salesperson**, **puzzle-solving**, **subset sum**, and **exam scheduling**
- Identify whether a function family is **tractable** or **intractable**
- Define the complexity classes **P** and **NP** and explain why they are important
- Identify whether a known algorithm runs in **P** and/or **NP** based on its runtime
- Use **heuristics** to find good-enough solutions to NP problems in polynomial time