CS 300 (Java) Modules 7–9 Cheat Sheet

■■ Module 7: Algorithm Analysis

Measure how efficient algorithms are in time (speed) and space (memory).

Complexity	Example	Description
O(1)	Accessing an array element	Constant time
O(log n)	Binary search	Cuts problem in half
O(n)	Linear search	Scans all elements
O(n log n)	Merge sort, quicksort	Efficient sorting
O(n^2)	Nested loops, bubble sort	Quadratic growth
O(2■)	Recursive brute force	Exponential growth

■ Module 8: Recursion

Recursion is when a method calls itself to solve smaller subproblems.

Structure: Base case (stops recursion) + Recursive case (calls itself).
Example:

```
int factorial(int n) {
if (n == 0) return 1;
return n * factorial(n-1);
}
```

Common recursive problems: Factorial, Fibonacci, Binary Search, Merge Sort, Towers of Hanoi.

■ Module 9: Sorting & Searching

Algorithm	Description	Time Complexity	Stable	Notes
Linear Search	Checks each element	O(n)	–	Unsorted data
Binary Search	Divides sorted array	O(log n)	–	Requires sorted input
Selection Sort	Finds smallest each pass	O(n^2)	No	Simple but slow
Insertion Sort	Inserts in order	O(n^2)	Yes	Good for small lists
Bubble Sort	Swaps adjacent out-of-order	O(n^2)	Yes	Easy to code
Merge Sort	Divide and merge halves	O(n log n)	Yes	Recursive, uses extra space
Quick Sort	Partition around pivot	O(n log n) avg	No	Recursive, fast on avg

■ Big Takeaways

- Use Big-O to compare algorithm growth rates.
- Recursion breaks problems into smaller self-similar parts.
- Efficient sorting/searching is crucial for handling large data sets.
- Always ensure recursion has a base case to avoid infinite calls.