CS3230 Midterm Cheatsheet

This cheatsheet covers key topics for the CS3230 midterm, including randomized algorithms, sorting properties, core algorithmic paradigms, and useful theorems. Use this as a quick reference during revision.

---

1. Randomized Algorithms

Definition: Algorithms that make random choices during execution.

- **Las Vegas algorithms** always produce a correct result; runtime is random (e.g., Randomized QuickSort).
- **Monte Carlo algorithms** have deterministic runtime; may produce incorrect results with small probability (e.g., randomized primality tests).

Key Concepts:

- **Expected Running Time**: (E[T]) computed over random choices.
- **Error Probability**: ensure (P[\text] \le \delta).
- **Amplification**: repeat trials to reduce error (e.g., majority vote in Monte Carlo).

Example: Randomized QuickSort

```
Expected time: O(n log n)
Worst-case time: O(n^2) (very unlikely if pivot is random)
```

---

2. Sorting Algorithm Properties

Algorithm	Best Case	Average Case	Worst Case	Stable	In-Place	Extra Space
Bubble Sort	O(n)	O(n^2)	O(n^2)	Yes	Yes	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	Yes	Yes	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	No	Yes	O(1)
Merge Sort	O(n log n)	O(n log n)	O(n log n)	Yes	No	O(n)
QuickSort	O(n log n)	O(n log n)	O(n^2)	No	Yes	O(log n)
HeapSort	O(n log n)	O(n log n)	O(n log n)	No	Yes	O(1)
Counting Sort	O(n + k)	O(n + k)	O(n + k)	Yes	No	O(k)
Radix Sort	O(nk)	O(nk)	O(nk)	Yes	No	O(n + k)

- **Stable**: preserves relative order of equal keys.
- **In-Place**: uses O(1) or O(log n) extra space.

---

3. Algorithmic Paradigms

3.1 Divide & Conquer

- **Pattern**: divide problem into subproblems, conquer each, combine results.
- **Typical Recurrence**: (T(n) = a,T(n/b) + f(n)).
- **Master Theorem** (for (a \ge 1, b > 1)):
    1. If (f(n) = O(n^{\log_b a - \epsilon})), then (T(n) = \Theta(n^{\log_b a})).
    2. If (f(n) = \Theta(n^{\log_b a} \log^k n)), then (T(n) = \Theta(n^{\log_b a} \log^{k+1} n)).
    3. If (f(n) = \Omega(n^{\log_b a + \epsilon})) and regularity holds, then (T(n) = \Theta(f(n))).

3.2 Greedy Algorithms

- Build solution in steps, choosing the locally optimal choice.
- **Examples**:
    - Activity Selection (interval scheduling)
    - Huffman Coding (optimal prefix codes)
    - Minimum Spanning Trees: Kruskal's & Prim's

Check: Greedy-choice property & optimal substructure.

3.3 Dynamic Programming (DP)

- **Principle**: optimal substructure + overlapping subproblems.
- **Techniques**: Memoization (top-down) or Tabulation (bottom-up).

Common DP Problems:

- Fibonacci numbers, Knapsack (0/1 & unbounded)
- Longest Common Subsequence (LCS)
- Matrix Chain Multiplication
- Coin Change

DP Steps:

1. Define subproblems.
2. Write recurrence.
3. Base cases.
4. Implement memo/tabulation.

---

4. Graph Algorithms

Algorithm	Purpose	Time Complexity
BFS	Shortest unweighted path	O(V + E)

Algorithm	Purpose	Time Complexity
DFS	Explore graph	O(V + E)
Dijkstra	Shortest weighted path	O((V + E) log V)
Bellman-Ford	Shortest path with neg. wts	O(VE)
Floyd-Warshall	All-pairs shortest paths	O(V^3)
Kruskal	Minimum spanning tree	O(E log E)
Prim	Minimum spanning tree	O(E + V log V)

- **Topological Sort**: DAG ordering via DFS or in-degree queue.
- **Strongly Connected Components**: Kosaraju's or Tarjan's.

---

5. Hashing

- **Hash Table Operations**: Insert, Delete, Search: average O(1).
- **Load Factor**: (\alpha = n/m) (elements/buckets).
- **Collision Resolution**:
  - Chaining: linked lists per bucket.
  - Open addressing: linear/quadratic probing, double hashing.

---

6. NP-Completeness & Approximation

- **Decision vs. Optimization** problems.
- **NP**: verifiable in polynomial time.
- **NP-Complete**: hardest in NP.
- **Reductions**: transform one problem to another.
- **Approximation Algorithms**:
  - Metric TSP: 2-approx via MST doubling.
  - Vertex Cover: 2-approx via matching.

> ***Tip**: Memorize key complexities and patterns; practice proving recurrences and verifying greedy correctness.*

Last updated: April 26, 2025