Stuff to know

Data Structures

1. Contiguous Arrays
    - Good and Bad vs Linked Lists
2. Linked Lists
    - Good and bad vs Arrays
    - Operations: search, insert, remove
3. Stack
    - Operations: push, pop
4. Queue
    - Operations: enqueue, dequeue
5. Dictionary (key-value storage)
    - Operations: search_by_key, insert, remove_by_key
    - Specialized Operations: max, min, predecessor, successor
6. Binary Search Tree
    - Operations: search, min, max, traversal, insert, delete (tricky)
7. Balanced Binary Search Tree
8. Priority Queue
    - Operations: insert, min/max, pop min/max.
9. Hash
    - Linked list vs linear probing implementations
    - Useful for duplicate detection (or substring detection).
10. Heap
    - Array representation vs binary tree representation
    -

Sorting

Algorithms:

1. Selection sort
    - Iterate along array
    - Move minimum of elements after i to i.

2. Heap sort
    - Selection sort into a heap
    - Convert array to min heap
    - Removing minimum is now O(log n)
    - Fast heap construction
3. Insert sort (Keeping sorted sublist)
    - Iterate along the array
    - For each element, put in the correct place up to i.
    - Incremental sort is online algorithm
4. Tree sort
    - Insert sort into a balanced binary tree
    - Recover sorted array as in order traversal
5. Merge sort (Divide and Conquer)
    - Good for sorting linked lists because does not require fast random access
    - Bad: needs buffer
6. Quick sort (Randomization)
    - O(n * h) where h is the height of the recursive call stack
    - To ensure random start state, randomly permute array before starting
    - Nut-bolt problem: given n nuts and n bolts, match each bolt to each nut in average O(nlogn).
    - Randomly pick a bolt, sort nuts based on bolt, then use the nut found to sort bolts.
7. Bucket sort
8. External sort (Multiway Mergesort)
    - Given k sorted lists on disk and k sorted top blocks in memory.
    - Make a heap with the top elements of the k top blocks.
    - Pop from heap and store in output array in memory and push from that subarray
    - When output array full, write to disk and when top block empty, read from disk.
9. Binary search
    - Modified binary search to count occurrences of a letter
    - One sided binary search

Useful to solve problems:

1. Binary Search

2. Closest Pair
3. Uniqueness
4. Looking for kth most occuring element
5. Selecting kth largest element
6. Convex hull

Graphs

Basics

1. Undirected v. directed
2. Weighted v. unweighted
3. Simple - no self loops, no multiedges
4. Spare v. dense
5. Cyclic v. acyclic

Implementation

1. Adjacency Matrix
2. Adjacency List
3. Graph traversal
    - 3 states: undiscovered, discovered, processed (visited all neighbors)
    - BFS
        - Finding shortest path by keeping a parent array while BFS and backtracking
        - Connected Component and bipartite Graphs
    - DFS
        - Finding cycles
        - Disconnecting graphs
    - Topological sorting
    - Strongly Connected

Combinatorial Search

Basic Implementation

```
def backtrack:
    if solution:
        process_solution
    else:
        construct_candidate_next_values_array
```

```
for each candidate in next_values:
    make_move
    backtrack
    unmake_move
```

Useful for

1. Generating powerset
2. Generating permutations
3. Generating all paths
4. Sudoku
    - Optimization: look for most constrained (least number of options) next square and look ahead to get possible values

Heuristic Search

Basic Examples

1. Random search
2. Local search - generate transition function that changes a few parameters of the search space slightly to generate new candidate to test
3. Simulated Annealing - local search with random jump to a far away solution. Prevents being trapped in local minima.
4. Genetic algorithm

Applications

1. Maximum cut problem - partitioning weighted graph G into sets with maximum weighted edges.

Dynamic Programming

Basics

1. Find recursive solution first
2. Consider whether you are reusing values. If so, precompute and store.

Examples

1. Fibonacci
2. Binomial Coefficient
3. Approximate String Matching (edit distance)
    - Assign cost to substitution, insertion, deletion.
    - Recursively compute min cost of string by taking minimum cost of sub, insert, delete.
    - Precompute minimum cost in a table
    - Reconstruct path
4. Substring Matching
5. Longest Common subsequence
6. Maximum Monotone subsequence
7. The partition problem without rearrangement