Pending's Bucket: Data Structures and Algorithms (DSA - 2)

Theory

This section covers theoretical concepts related to data structures and algorithms that need to be studied or revisited for better understanding.

- **Data Structures**
  - **Stack**: Stack overflow vs. underflow, purpose of stack pointer, applications (e.g., undo-redo operations), monotonic stack
  - **Queue**: Types of queues (circular queue, priority queue, double-ended queue, bounded queues), applications of circular and double-ended queues
  - **Hash Tables**: Hash table vs. hash set, load factor, open addressing, separate chaining, linear probing, quadratic probing, double hashing, rehashing, collision handling, hashing vs. encryption, popular hashing algorithms (SHA1, MD5, CRC32), time complexity of hash table operations
  - **Arrays**: Homogeneous vs. heterogeneous arrays, jagged arrays, array vs. hash table, stack vs. array
- **Algorithms**
  - **Sorting Algorithms**: Bubble sort, insertion sort, selection sort, merge sort, quick sort, heap sort, in-place sorting, stable sorting, time and space complexity of sorting algorithms, pivot selection in quick sort, why merge sort is preferred for linked lists, disadvantages of merge sort and quick sort, why bubble sort is stable, best sorting algorithm for partially sorted or small arrays
  - **Divide and Conquer**: Concept and applications
  - **Backtracking**: Basic concepts
  - **Sliding Window**: Pattern and applications
  - **Complexity Analysis**: Time and space complexity calculations, Big-O, Big-Theta, Big-Omega notations, logarithmic functions (log n vs. n log n), reasons for O(nš) worst-case in quick sort
  - **Hashing**: Hash functions, collision resolution methods, perfect hash function, applications of hash tables (e.g., database indexing)

- **Strings**
  - String immutability in languages like Python
  - Character encoding: UTF-8, size of a character in UTF-8, control characters
- **Miscellaneous**
  - Linear vs. non-linear data structures
  - B-tree concepts
  - Kilobyte vs. kibibyte

Practicals

This section lists practical exercises and coding tasks to be implemented or practiced, focusing on data structures and algorithms.

- **Stack Operations**
  - Implement stack using linked list
  - Implement stack using queue
  - Reverse a string using stack (e.g., "HELLO WORLD" to "OLLEH DLROW")
  - Reverse a stack (iterative and recursive)
  - Sort a stack using a temporary stack
  - Delete a specific node from a stack
  - Delete the middle element of a stack
  - Implement a stack that rejects duplicate values
  - Implement a stack with push, pop, and get current highest number in O(1) complexity
  - Check if a string is a palindrome using a stack
  - Valid parentheses problem (LeetCode 20, including modified version to count invalid pairs)
- **Queue Operations**
  - Implement queue using stack
  - Implement circular queue (with max length)
  - Implement double-ended queue using linked list
  - Reverse a queue
  - Enqueue, dequeue, and display operations
  - Implement circular buffers
- **Hash Table Operations**

- Implement hash table with collision handling (chaining with linked list, open addressing)
- Implement linear probing, quadratic probing, and double hashing
- Rehashing implementation
- Find the first non-repeating character in a string using a hash table (e.g., "swiss")
- Count the frequency of characters in a string (e.g., "Mississippi") using a hash table
- Remove duplicates from a string using a hash table
- Find two numbers in an array that add up to a target sum (Two Sum, LeetCode 1)
- Find the least occurred number in a string using a hash table
- Find uncommon elements from two arrays using a hash table
- Check if a string contains duplicates using a hash table

- **Sorting Algorithms**
  - Implement bubble sort, insertion sort, selection sort, merge sort, quick sort
  - Sort a string using merge sort
  - Merge two sorted arrays into a single sorted array in O(n) time
  - Quick sort without additional arrays
  - Sort an array of objects based on a property (e.g., .amount)
  - Sort an array of students based on age
  - Check if an array is sorted with linear time complexity
- **String Operations**
  - Find the first missing number in an array
  - Find the subarray with the maximum sum (Kadanes algorithm)
  - Find common characters from two strings
  - Find the longest consecutive repeating characters in a string
  - Find the second longest word in a sentence
  - Convert a string like "APPLE" to "A-pp-ppp-llll-eeeee"
  - Convert the first character of a string to uppercase
  - Build a URL from a base URL and query parameters passed as a dictionary
  - Valid anagram problem (compare two dictionaries, handle corner cases)
- **LeetCode Blind 75 Practice**
  - Two Sum (1)

  - Merge Two Sorted Lists (21)
  - Valid Parentheses (20)
  - Find the first missing number in an array
  - Find the subarray with the maximum sum (Kadanes algorithm)
- **Other**
  - Remove odd-indexed elements from an array
  - Find duplicate students in an array
  - Move zeroes to the end of an array

Results

The following summarizes the status of the pending topics and tasks:

- **Completed Topics**: Some problems like merge two sorted lists and valid parentheses have been attempted, but solutions often took more time or failed to handle corner cases correctly.
- **Partially Completed**:
  - Quick sort implementation (needs optimization)
  - Valid anagram problem (second half logic and corner cases incomplete)
  - Hash table implementation (needs improvement in collision handling)
  - Merge sort implementation (requires further clarity)
  - Circular queue implementation (needs more practice)
- **Pending Tasks**:
  - Deepen understanding of sorting algorithms (bubble, insertion, selection, merge, quick, heap) and their complexities
  - Master collision handling in hash tables (chaining, open addressing, linear/quadratic probing, double hashing)
  - Implement and optimize stack and queue conversions (stack using queue, queue using stack)
  - Practice Blind 75 LeetCode problems, focusing on brute force and optimal solutions (refer to NeetCode.io and YouTube)
  - Revise concepts like load factor, rehashing, and perfect hash functions
  - Complete circular queue and double-ended queue implementations
  - Improve logic-building for array and string problems
  - Optimize code for problems like Two Sum and Valid Parentheses to achieve O(n) complexity

- **Recommendations**:
  - Refer to resources like NeetCode.io, LeetCode, and YouTube for optimal solutions.
  - Practice debugging and understanding problems before implementation.
  - Focus on mastering time and space complexity analysis.
  - Revisit theoretical concepts regularly to ensure clarity.

– Explore GitHub repositories (e.g., Ajaymanikandan0x/dsa$_in_dart$, $RahulR9809/Ds)for reference imp$