🚀 PFRDA Grade A IT Officer - Last-Minute Quick Reference

⚡ Algorithm Complexity Cheat Sheet

Sorting Algorithms

Algorithm	Best	Average	Worst	Space	Stable
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Yes
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	No
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Yes
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)	Yes
Quick Sort	O(nlogn)	O(nlogn)	$O(n^2)$	O(logn)	No
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(1)	No
Counting Sort	O(n+k)	O(n+k)	O(n+k)	O(k)	Yes
Radix Sort	O(d×n)	O(d×n)	O(d×n)	O(n+k)	Yes

Graph Algorithms

Algorithm	Time Complexity	Space	Use Case
BFS	O(V + E)	O(V)	Shortest path (unweighted)
DFS	O(V + E)	O(V)	Connected components
Dijkstra	O((V+E) log V)	O(V)	Shortest path (non-negative)
Bellman-Ford	O(VE)	O(V)	Shortest path (with negative)
Floyd-Warshall	$O(V^3)$	$O(V^2)$	All pairs shortest path
Kruskal's MST	O(E log E)	O(V)	Minimum spanning tree
Prim's MST	O((V+E) log V)	O(V)	Minimum spanning tree
Topological Sort	O(V + E)	O(V)	DAG ordering

🌳 Tree Traversal Mnemonics

Remember the Order:

- **Inorder**: Left → **Root** → Right (gives sorted order in BST)
- **Preorder**: **Root** → Left → Right (good for copying tree)
- **Postorder**: Left → Right → **Root** (good for deleting tree)
- **Level Order**: Use **Queue** (BFS approach)

Tree Construction Rules:

1. **Preorder + Inorder** → Can construct unique tree
2. **Postorder + Inorder** → Can construct unique tree
3. **Preorder + Postorder** → Cannot construct unique tree (except full binary tree)

🔍 Search Algorithm Decision Tree

```
Is data SORTED?
├── YES → Use Binary Search O(log n)
└── NO → Use Linear Search O(n)

For BST:
├── Search → O(log n) average, O(n) worst
├── Insert → O(log n) average, O(n) worst
└── Delete → O(log n) average, O(n) worst

For Hash Table:
├── Search → O(1) average, O(n) worst
├── Insert → O(1) average, O(n) worst
└── Delete → O(1) average, O(n) worst
```

🎯 Dynamic Programming Patterns

Classic DP Problems - Remember These!

1. 0/1 Knapsack

```python
dp[i][w] = max(dp[i-1][w], values[i-1] + dp[i-1][w-weights[i-1]])
```

2. Longest Common Subsequence (LCS)

```python
if str1[i-1] == str2[j-1]:
    dp[i][j] = dp[i-1][j-1] + 1
else:
    dp[i][j] = max(dp[i-1][j], dp[i][j-1])
```

3. Coin Change (Minimum coins)

```python
dp[amount] = min(dp[amount], dp[amount-coin] + 1) for all coins
```

4. Edit Distance

```python
if s1[i-1] == s2[j-1]:
    dp[i][j] = dp[i-1][j-1]
else:
    dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])
```

🔄 Graph Algorithms Quick Implementation

DFS Template (Recursive)

```python
def dfs(graph, node, visited):
    visited[node] = True
    print(node)
    for neighbor in graph[node]:
        if not visited[neighbor]:
            dfs(graph, neighbor, visited)
```

BFS Template

```python
from collections import deque

def bfs(graph, start):
    visited = set([start])
    queue = deque([start])
    
    while queue:
        node = queue.popleft()
        print(node)
        for neighbor in graph[node]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)
```

Dijkstra's Algorithm Template

```python
import heapq

def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    pq = [(0, start)]
    
    while pq:
        current_distance, node = heapq.heappop(pq)
        
        if current_distance > distances[node]:
            continue
        
        for neighbor, weight in graph[node]:
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(pq, (distance, neighbor))
```

⚖️ Algorithm Selection Guide

When to Use Each Algorithm:

Sorting Decision Matrix:

- **Need stability?** → Merge Sort, Counting Sort
- **Memory limited?** → Heap Sort, In-place Quick Sort
- **Nearly sorted data?** → Insertion Sort
- **Known range of integers?** → Counting Sort
- **General purpose?** → Quick Sort (randomized)
- **Guaranteed O(n log n)?** → Merge Sort, Heap Sort

Graph Algorithm Selection:

- **Unweighted shortest path?** → BFS
- **Connected components?** → DFS
- **Single source shortest path (non-negative weights)?** → Dijkstra

- **Single source with negative weights?** → Bellman-Ford
- **All pairs shortest path?** → Floyd-Warshall
- **Minimum spanning tree?** → Kruskal's (sparse), Prim's (dense)

Design Technique Selection:

- **Optimal substructure + overlapping subproblems?** → Dynamic Programming
- **Locally optimal → globally optimal?** → Greedy Algorithm
- **Divide into similar subproblems?** → Divide and Conquer
- **Try all possibilities?** → Backtracking

🧮 Master Theorem Quick Reference

For recurrence: **T(n) = aT(n/b) + f(n)**

- **Case 1:** f(n) = O(n^c) where c < log_b(a) → **T(n) = Θ(n^(log_b(a)))**
- **Case 2:** f(n) = Θ(n^c × log^k(n)) where c = log_b(a) → **T(n) = Θ(n^c × log^(k+1)(n))**
- **Case 3:** f(n) = Ω(n^c) where c > log_b(a) → **T(n) = Θ(f(n))**

Common Examples:

- **Merge Sort:** T(n) = 2T(n/2) + O(n) → **O(n log n)**
- **Binary Search:** T(n) = T(n/2) + O(1) → **O(log n)**
- **Strassen's:** T(n) = 7T(n/2) + O(n^2) → **O(n^2.81)**

🏆 Most Frequently Asked PFRDA Questions

Question Type 1: Algorithm Identification

Example: "Which algorithm is used to find strongly connected components?" **Answer:** Kosaraju's Algorithm or Tarjan's Algorithm

Question Type 2: Complexity Analysis

Example: "What is the time complexity of building a heap from n elements?" **Answer:** O(n) - not O(n log n)!

Question Type 3: Data Structure Selection

Example: "Best data structure for implementing LRU cache?" **Answer:** Hash Map + Doubly Linked List

Question Type 4: Tree Construction

Example: "Can we construct a unique binary tree from preorder and postorder?" **Answer:** No (except for full binary trees)

Question Type 5: Graph Properties

Example: "Minimum edges needed to connect n vertices?" **Answer:** n-1 (spanning tree property)

🎯 Exam Day Strategy

Time Allocation (for 2-hour exam):

- **Tree/Graph Questions (40 marks):** 45 minutes
- **Sorting/Searching (20 marks):** 25 minutes
- **DP/Design Techniques (25 marks):** 35 minutes
- **Review and difficult questions:** 15 minutes

Question Solving Approach:

1. **Read question completely** - identify keywords
2. **Classify problem type** - tree, graph, optimization, etc.
3. **Recall template** - use memorized patterns
4. **Check edge cases** - empty input, single element
5. **Verify complexity** - does it match constraints?

💡 Last-Minute Memory Tricks

Acronyms to Remember:

- **BFS uses QUEUE** → "**B**readth **Q**ueue"
- **DFS uses STACK** → "**D**epth **S**tack"
- **Dijkstra for Non-Negative** → "**D**ijkstra **N**o **N**egative"
- **Bellman-Ford for Negative** → "**B**ellman **N**egative"

Visual Memory Aids:

- **MST algorithms:** Kruskal = **K**rusty crab (sort edges), Prim = **P**riority queue
- **Sorting stability: MIRC** - Merge, Insertion, Radix, Counting are stable
- **Tree height:** Balanced = O(log n), Skewed = O(n)

⚡ Speed Calculation Tricks

For Binary Operations:

- **$2^{10} \approx 1000$** (exactly 1024)
- **$\log_2(1000) \approx 10$**
- **$\log_2(1{,}000{,}000) \approx 20$**

Hash Table Load Factor:

- **$\alpha = n/m$** (elements/table size)
- **Good performance:** $\alpha \leq 0.7$
- **Expected probes:** $1/(1-\alpha)$ for open addressing

Tree Properties:

- **Complete binary tree height:** $\lfloor \log_2(n) \rfloor$
- **Max nodes at level i:** 2^i
- **Min nodes for height h:** $h+1$
- **Max nodes for height h:** $2^{(h+1)} - 1$

🎪 Tricky Questions to Watch Out For

Common Pitfalls:

1. **Heap construction is O(n), not O(n log n)**
2. **Quick Sort worst case is $O(n^2)$**
3. **Binary Search needs SORTED array**
4. **DFS can be implemented iteratively too**
5. **MST has exactly n-1 edges for n vertices**

Edge Cases Always Test:

- Empty input (n = 0)
- Single element (n = 1)
- Already sorted array
- Reverse sorted array
- All elements same
- Graph with no edges

- Disconnected graph

🔥 Final Confidence Boosters

If You See These Keywords:

- **"Minimum/Maximum"** → Think DP or Greedy
- **"Path between nodes"** → Think BFS/DFS
- **"Shortest path"** → Think Dijkstra/BFS
- **"All pairs"** → Think Floyd-Warshall
- **"Sorted order"** → Think Binary Search/BST
- **"Connected components"** → Think DFS/Union-Find

Remember Success Formula:

Understanding + Practice + Confidence = PFRDA Success!

🎯 Final Checklist (Day Before Exam)

- [ ] Reviewed all tree traversal orders
- [ ] Memorized complexity table for sorting algorithms
- [ ] Practiced BFS and DFS implementations
- [ ] Remembered Dijkstra's algorithm steps
- [ ] Understood MST algorithm differences
- [ ] Reviewed DP problem patterns
- [ ] Prepared mentally for 2-hour focused exam
- [ ] Got good sleep and planned exam day logistics

🌟 You're Ready! Trust Your Preparation!

Remember: The PFRDA exam tests your understanding of fundamental concepts. Stay calm, think logically, and apply the patterns you've learned. **You've got this!** 🚀

Best of luck for your PFRDA Grade A Assistant Manager IT Officer exam! 🎊