**✅ Core Data Structure Interview Questions and Answers**

**1. What is the difference between Array and LinkedList?**

| **Feature** | **Array** | **LinkedList** |
| --- | --- | --- |
| Memory | Contiguous | Non-contiguous |
| Access | O(1) for index access | O(n) traversal |
| Insertion/Del | Costly at middle positions | Efficient for insert/delete |

**Follow-up:**  
Q: How does cache locality affect Array performance?  
A: Arrays are cache-friendly due to contiguous memory; LinkedLists are not.

**2. What is the time complexity of common operations in data structures?**

| **Data Structure** | **Access** | **Search** | **Insertion** | **Deletion** |
| --- | --- | --- | --- | --- |
| Array | O(1) | O(n) | O(n) | O(n) |
| Linked List | O(n) | O(n) | O(1) | O(1) |
| HashMap | O(1) | O(1) | O(1) | O(1) |
| BST | O(log n) | O(log n) | O(log n) | O(log n) |
| Heap | N/A | O(n) | O(log n) | O(log n) |

**3. How is HashMap implemented in Java?**

* Uses an **array of buckets** (Node<K,V>[]).
* Each bucket is a **linked list** or **tree (after threshold)**.
* Uses hashCode() and equals() to store and retrieve keys.

java

CopyEdit

map.put("John", 45); // internally:

int hash = key.hashCode();

int index = hash % capacity;

**Follow-up:** What happens if two keys have the same hash?  
A: Collision happens → values are stored in linked list or tree (Java 8+).

**4. What is the difference between Stack and Queue?**

* **Stack**: LIFO (Last In First Out)
  + push(), pop()
* **Queue**: FIFO (First In First Out)
  + enqueue(), dequeue()

**Follow-up:** Where is Stack used in JVM?  
A: JVM uses stack memory for method calls and local variables.

**✅ Intermediate-Level Questions**

**5. How does a Binary Search Tree (BST) work?**

* Each node has left < root < right.
* In-order traversal gives sorted order.

java

CopyEdit

class Node {

int data;

Node left, right;

}

**Follow-up:** What if BST becomes unbalanced?  
A: It becomes a linked list → O(n). Use AVL or Red-Black Tree.

**6. How do you detect a cycle in a Linked List?**

**Answer:** Use Floyd’s Cycle Detection (Tortoise and Hare).

java

CopyEdit

Node slow = head, fast = head;

while(fast != null && fast.next != null) {

slow = slow.next;

fast = fast.next.next;

if(slow == fast) return true;

}

return false;

**7. What is a Trie? Use case?**

* **Prefix tree** used for efficient string lookups (e.g. dictionary, autocomplete).
* Time Complexity: O(m), where m = length of word.

**8. What is a Heap and where is it used?**

* Binary Heap: Complete binary tree with heap property.
* **MinHeap**: Parent ≤ child
* **MaxHeap**: Parent ≥ child

**Applications:**

* Priority Queue
* Dijkstra’s algorithm
* Heap sort

**✅ Advanced Data Structure Questions (With Real-World Use)**

**9. Design a data structure to get O(1) for insert, delete, and getRandom().**

Use HashMap + ArrayList.

java

CopyEdit

Map<Integer, Integer> map; // val -> index

List<Integer> list;

void insert(int val) {

if (!map.containsKey(val)) {

map.put(val, list.size());

list.add(val);

}

}

void remove(int val) {

if (map.containsKey(val)) {

int index = map.get(val);

int last = list.get(list.size()-1);

list.set(index, last);

map.put(last, index);

list.remove(list.size()-1);

map.remove(val);

}

}

int getRandom() {

return list.get(random.nextInt(list.size()));

}

**10. What is the difference between TreeMap and HashMap?**

| **Feature** | **HashMap** | **TreeMap** |
| --- | --- | --- |
| Ordering | No order | Sorted by key |
| Performance | O(1) | O(log n) |
| Null Keys | 1 allowed | Not allowed |

**11. Design LRU Cache.**

Use LinkedHashMap or custom LinkedList + HashMap.

java

CopyEdit

class LRUCache extends LinkedHashMap<Integer, Integer> {

int capacity;

LRUCache(int cap) {

super(cap, 0.75f, true);

this.capacity = cap;

}

protected boolean removeEldestEntry(Map.Entry<Integer, Integer> eldest) {

return size() > capacity;

}

}

**12. What is Union-Find / Disjoint Set Union (DSU)?**

Used for connectivity in graphs (e.g., Kruskal’s algorithm).

* find(x) → returns root
* union(x, y) → connects sets

Uses **path compression** and **union by rank** for efficiency.

**✅ Real-world Scenario Questions**

**13. You are designing an autocomplete feature. Which data structure would you use?**

* **Trie** to store words efficiently.
* Each node stores prefix path and a list of suggestions (optional).

**14. How would you store a stream of median values efficiently?**

Use **Two Heaps**:

* MaxHeap for lower half
* MinHeap for upper half

Median = root of maxHeap (odd) or avg of both (even).

**15. Explain Bloom Filter.**

* Probabilistic data structure.
* Checks if an element **might be present** or **definitely not present**.
* Fast, space-efficient → used in databases, caches (e.g., Redis).

**📘 Want More?**

Would you like:

* ✅ PDF version of this set?
* ✅ 50–100 MCQs + Scenario questions (quiz format)?
* ✅ GitHub coding repo with common implementations?
* ✅ Diagram pack (Heap, Tree, Trie, Graph, DSU, etc.)?

Let me know which format you'd like!

**You said:**

50–100 MCQs + Scenario questions (quiz format)?

**ChatGPT said:**

Updated saved memory

Great! Here's a **sample pack of 20 Data Structure MCQs + 5 Scenario-based Questions** to get you started. I’ll follow up with the rest in batches if you’d like a full 50–100.

**📘 Multiple Choice Questions (MCQs) – *Data Structures***

**1. Which of the following has O(1) time complexity for lookup?**

A) LinkedList  
B) HashMap  
C) TreeMap  
D) PriorityQueue

✅ **Answer:** B) HashMap

**2. What is the worst-case time complexity of searching in a Binary Search Tree (BST)?**

A) O(log n)  
B) O(n)  
C) O(1)  
D) O(n log n)

✅ **Answer:** B) O(n) *(when the tree is skewed)*

**3. What is the time complexity of inserting an element into a heap?**

A) O(1)  
B) O(n)  
C) O(log n)  
D) O(n log n)

✅ **Answer:** C) O(log n)

**4. Which data structure is ideal for implementing undo functionality?**

A) Queue  
B) Stack  
C) Heap  
D) Array

✅ **Answer:** B) Stack

**5. In which case would you prefer a LinkedList over an ArrayList?**

A) Fast random access  
B) Frequent insertions and deletions  
C) Sorting large datasets  
D) When thread safety is required

✅ **Answer:** B) Frequent insertions and deletions

**6. Which traversal method is used to get a sorted list from BST?**

A) Pre-order  
B) Post-order  
C) In-order  
D) Level-order

✅ **Answer:** C) In-order

**7. Which data structure is used in recursion?**

A) Heap  
B) Queue  
C) Stack  
D) Tree

✅ **Answer:** C) Stack

**8. What is the output order of a Queue?**

A) LIFO  
B) FIFO  
C) Random  
D) Priority-based

✅ **Answer:** B) FIFO

**9. A priority queue can be efficiently implemented using:**

A) Stack  
B) Heap  
C) HashMap  
D) Graph

✅ **Answer:** B) Heap

**10. Which data structure is used in DFS of a graph?**

A) Queue  
B) Stack  
C) PriorityQueue  
D) Trie

✅ **Answer:** B) Stack

**11. What does a Trie optimize?**

A) Hashing  
B) Integer lookups  
C) Prefix matching  
D) Dynamic arrays

✅ **Answer:** C) Prefix matching

**12. What’s the space complexity of an adjacency matrix for a graph with V vertices?**

A) O(V)  
B) O(E)  
C) O(V²)  
D) O(log V)

✅ **Answer:** C) O(V²)

**13. What is the time complexity of HashMap’s get() in average case?**

A) O(n)  
B) O(log n)  
C) O(1)  
D) O(n log n)

✅ **Answer:** C) O(1)

**14. Which structure allows O(1) insertion and deletion at both ends?**

A) Array  
B) LinkedList  
C) Deque  
D) Stack

✅ **Answer:** C) Deque

**15. In a Min Heap, what is the parent of node at index i?**

A) (i-1)/2  
B) 2\*i + 1  
C) i+1  
D) i-1

✅ **Answer:** A) (i-1)/2

**16. Which is not true about HashSet?**

A) It allows duplicate keys  
B) It uses hashCode() internally  
C) It has O(1) add operation  
D) It stores only unique values

✅ **Answer:** A) It allows duplicate keys

**17. What is the best use-case of a circular queue?**

A) DFS  
B) Recursion  
C) CPU Scheduling  
D) Caching

✅ **Answer:** C) CPU Scheduling

**18. A data structure with enqueue and dequeue at both ends is called:**

A) Queue  
B) Deque  
C) Stack  
D) PriorityQueue

✅ **Answer:** B) Deque

**19. What is the amortized time for appending to a dynamic array (ArrayList)?**

A) O(1)  
B) O(log n)  
C) O(n)  
D) O(n²)

✅ **Answer:** A) O(1)

**20. Which structure is best for finding the longest common prefix of a set of strings?**

A) Array  
B) LinkedList  
C) HashMap  
D) Trie

✅ **Answer:** D) Trie

**📘 Scenario-Based Questions (Real-World Applications)**

**21. You need to design a cache that evicts least recently used elements. Which structure will you use?**

✅ **Answer:** Use a combination of HashMap + Doubly LinkedList (or LinkedHashMap in Java)

**22. You have a stream of numbers and need to return the median at any point. What approach do you use?**

✅ **Answer:** Two Heaps

* MaxHeap for lower half
* MinHeap for upper half

**23. You are asked to implement an autocomplete system. Which data structure would help the most?**

✅ **Answer:** Trie (Prefix Tree)

**24. Your service handles millions of queries for URL blacklist check. Which DS gives fast space-efficient lookups with acceptable false positives?**

✅ **Answer:** Bloom Filter

**25. You need to evaluate an arithmetic expression with brackets. What structure should you use?**

✅ **Answer:** Stack (to track opening and closing brackets)

Would you like the next **25, 50, or all 75+** delivered:

* 📄 As **PDF**
* 🧠 As **Mock quiz app (with score tracking)**
* 💻 As **Web-based React quiz app**
* 📂 As **GitHub repo with interactive solutions**

Let me know your preferred format and delivery mode!

**You said:**

cross questions for experience

**ChatGPT said:**

Here are **30+ Cross/Follow-up Questions** for experienced professionals on **Data Structures**, including **advanced**, **design-level**, and **optimization-based** scenarios. These are typically asked after you give a correct or partially correct answer in interviews, especially for **8–15+ years of experience**.

**🔁 HashMap / Hashing – Cross Questions**

**Q1:**

You said HashMap gives O(1) access. What happens if all keys hash to the same bucket?

✅ Cross Answer:  
If all keys hash to the same bucket, the bucket becomes a linked list (or a balanced tree in Java 8+). This degrades performance from **O(1)** to **O(n)** or **O(log n)** depending on the collision handling strategy.

**Q2:**

How does Java HashMap handle collision internally?

✅ Cross Answer:  
Initially via **chaining** using a linked list. From Java 8+, if the list exceeds a threshold (usually 8), it is converted to a **red-black tree** for better lookup performance.

**Q3:**

Can you store null keys and values in a HashMap?

✅ Cross Answer:  
Yes, **one null key** is allowed. Multiple **null values** are allowed. This is because null.hashCode() will throw NPE, so HashMap has a special condition to handle key == null.

**📚 Tree / BST / AVL / Trie – Cross Questions**

**Q4:**

You mentioned BST has O(log n) operations. Is it guaranteed?

✅ Cross Answer:  
No. It's only guaranteed if the tree is **balanced**. In the worst case (skewed BST), the time complexity is **O(n)**.

**Q5:**

Why would you use a Trie over a HashMap for prefix search?

✅ Cross Answer:  
HashMap doesn't guarantee **prefix ordering**, while Trie allows **shared prefix storage**, leading to **faster lookups** and **space optimization** for large prefix-based datasets.

**Q6:**

Can a Trie be memory-efficient? Isn’t it memory-heavy?

✅ Cross Answer:  
Tries **consume more memory** than HashMaps but can be optimized using:

* **Compressed Tries / Radix Trees**
* Using **arrays instead of HashMaps** when character set is limited
* **Bitmask-based** nodes

**📥 Heap / PriorityQueue – Cross Questions**

**Q7:**

You said Heapsort is O(n log n). Why don’t we use it over MergeSort or QuickSort?

✅ Cross Answer:  
Because Heapsort is **not stable** and performs more **data swaps**, leading to poor **cache locality**. MergeSort is preferred when stability matters. QuickSort performs better in practice due to average-case O(n log n) and cache friendliness.

**Q8:**

How is a PriorityQueue different from a normal Queue?

✅ Cross Answer:  
PriorityQueue retrieves elements based on **priority**, not insertion order. Internally implemented as **MinHeap** or **MaxHeap** depending on comparator.

**🔄 LinkedList / ArrayList / Stack / Queue – Cross Questions**

**Q9:**

You chose LinkedList for insertions. Is it always better than ArrayList?

✅ Cross Answer:  
Not always. While insertions in the middle are faster in LinkedList (O(1)), **traversal in LinkedList is O(n)**, and it has **higher memory overhead** due to node pointers. ArrayList offers **better cache locality** and **faster random access**.

**Q10:**

How would you implement a queue using stacks?

✅ Cross Answer:  
Use two stacks:

* inStack for push
* outStack for pop  
  Transfer inStack → outStack when popping and outStack is empty.

**🎯 Graph / DSU / Adjacency Matrix – Cross Questions**

**Q11:**

You chose adjacency matrix for graph storage. Is it optimal?

✅ Cross Answer:  
Only for **dense graphs**. For **sparse graphs**, **adjacency list** is preferred as it saves space (O(V + E) vs O(V²)).

**Q12:**

How is Union-Find (Disjoint Set) optimized?

✅ Cross Answer:  
Two optimizations:

* **Path Compression** → flatten tree during find
* **Union by Rank** → attach smaller tree to larger

Both result in near-constant amortized time: **O(α(n))**, where α is inverse Ackermann.

**🧠 Design-Based / System Thinking Cross Questions**

**Q13:**

You mentioned LRU Cache with LinkedHashMap. What are its limitations?

✅ Cross Answer:

* Not thread-safe
* Memory limits must be manually enforced
* Hard to customize eviction policies (e.g., LFU)

For better control, use **custom DoublyLinkedList + HashMap** or **Caffeine/Guava** libraries.

**Q14:**

Why not use HashSet instead of HashMap to track keys?

✅ Cross Answer:  
HashSet is a **wrapper over HashMap** that stores dummy values. HashMap gives you **value-based access** which HashSet does not.

**Q15:**

Which data structures would you use to design a leaderboard with top-K elements updating in real time?

✅ Cross Answer:

* **MinHeap** of size K
* Or a **Balanced BST** like TreeMap
* For frequent updates → use **Segment Tree** or **Binary Indexed Tree (Fenwick Tree)**

**⚙️ Concurrency & Performance Related**

**Q16:**

Is Java’s HashMap thread-safe?

✅ Cross Answer:  
No. For thread-safety, use:

* Collections.synchronizedMap(map) (synchronized wrapper)
* ConcurrentHashMap (recommended for high concurrency)

**Q17:**

Why is ConcurrentHashMap better than Hashtable?

✅ Cross Answer:

* **Segmented locks** or lock striping (Java 7)
* **Bucket-level synchronization** (Java 8+)
* **Finer-grained locking → better concurrency**

**📌 Misc + Real-Time Design Follow-Ups**

**Q18:**

Your design uses a Bloom Filter. How do you handle false positives?

✅ Cross Answer:

* Acceptable in many use cases (e.g., cache, URL blacklists)
* Confirm via **backup DB check** if needed
* Tune **hash functions and bit array size**

**Q19:**

How would you persist a Trie on disk for huge datasets?

✅ Cross Answer:

* Use **serialization**, or
* Convert Trie to a **DAWG (Directed Acyclic Word Graph)**
* Use **prefix encoding** or **compressed formats** like **Marisa Trie**

**Q20:**

In a distributed system, how would you maintain a distributed queue?

✅ Cross Answer:

* Use **Kafka**, **RabbitMQ**, or **Redis Streams**
* For custom DS: use **consistent hashing** + distributed locks (Zookeeper/Etcd)