Q1. [Definition] What is the core principle of a Stack?
A. First In First Out (FIFO)
B. Last In First Out (LIFO)
C. First Come First Serve
D. Random Access
Answer: B
Explanation: Stack operates on LIFO — the last inserted element is the first to be removed.

Q2. [Definition] Which Java data structure is traditionally used to implement a stack?
A. ArrayList
B. LinkedList
C. Stack
D. HashMap
Answer: C
Explanation: Java provides a Stack class that supports standard stack operations like push and pop.
Q3. [Definition] Which of the following is NOT a valid stack operation?
A. push()
B. pop()
C. peek()
D. addAtFront()
Answer: D
Explanation: addAtFront() is not a typical stack method; stack operations are limited to the top element.
Q4. [Definition] Which of these real-world problems is best solved using a stack?
A. Finding shortest path
B. Balancing parentheses
C. Sorting arrays
D. Counting inversions
Answer: B
Explanation: Matching and balancing symbols like parentheses is efficiently solved using a stack.
Q5. [Definition] What happens when you pop from an empty stack in Java?
A. Returns null
B. Returns 0
C. Throws EmptyStackException
D. Skips the operation
Answer: C
Explanation: Attempting to pop from an empty stack throws an EmptyStackException.
Q6. [Moderate] Which of the following strings is valid with balanced parentheses?
A. "(())"
B. "(()"
C. "())("
D. "(()))"
Answer: A
Explanation: A valid expression requires every opening bracket to be matched by a corresponding closing one in the correct order.
Q7. [Moderate] What is the key condition for a string with parentheses to be valid?
A. More closing brackets

B. Stack never becomes empty
C. Stack is empty after entire traversal
D. No brackets
Answer: C
Explanation: If the stack is empty after processing all characters, the string is valid.
Q8. [Moderate] What happens if you push every opening bracket and pop for matching closing in a valid parentheses check?
A. Brackets are reversed
B. You get extra opening brackets
C. Stack is empty at the end if valid
D. Stack overflows
Answer: C
Explanation: Balanced parentheses result in a completely emptied stack after matching.
Q9. [Moderate] In a valid parentheses problem, what condition causes early invalidation?
A. Encountering a] when stack top is [
B. Stack size greater than 0
C. Skipping characters
D. Ignoring closing brackets
Answer: A
Explanation: A mismatch in bracket types (e.g.,] vs [) indicates invalid parentheses immediately.
Q10. [Moderate] Which data structure is best for implementing a valid parentheses checker?
A. Queue
B. Stack
C. HashMap
D. Tree
Answer: B
Explanation: Stack helps in tracking unmatched opening brackets efficiently.
Q11. [Moderate] What is the purpose of a Min Stack?
A. Retrieve smallest element in O(n)
B. Find max value
C. Return minimum in O(1) time
D. Track all values inserted
Answer: C
Explanation: Min Stack allows retrieval of the minimum element in constant time.
Q12. [Moderate] Which of the following operations must be modified to maintain Min Stack functionality?
A. pop()
B. peek()
C. push()
D. Both A and C
Answer: D
Explanation: Both push() and pop() must be modified to track and maintain the current minimum.
Q13. [Moderate] How can you keep track of the minimum in a Min Stack?
A. Use a counter
B. Sort after every insert
C. Use an auxiliary stack for mins
D. Store in HashMap

Answer: C
Explanation: An auxiliary stack can store the current minimum alongside the main stack.
Q14. [Moderate] What value will getMin() return after pushing [3, 5, 2, 1] into a Min Stack?
A. 5
B. 3
C. 1
D. 2
Answer: C
Explanation: The minimum value in the stack is 1.
Q15. [Moderate] What should happen in pop() to correctly update the min value in Min Stack?
A. Remove from both main and min stack
B. Remove only from min stack
C. Clear entire stack
D. Reset min to zero
Answer: A
Explanation: To maintain accuracy, the pop operation must remove the top from both stacks.
Q1. [Definition] What is a monotonic stack?
A. A stack that stores elements in increasing or decreasing order
B. A queue that stores sorted elements
C. A binary search tree
D. A heap
Answer: A
Explanation: A monotonic stack is maintained in either strictly increasing or decreasing order based on the problem.
Q2. [Definition] In a monotonic decreasing stack, which element is at the top?
A. The largest so far
B. The smallest so far
C. The next greater
D. A random element
Answer: B
Explanation: A decreasing stack removes larger elements, so the top is always the smallest seen.
Q3. [Definition] What is the purpose of a monotonic stack in "Next Greater Element"?
A. Sort the array
B. Reduce the array size
C. Track elements in order for efficient comparison
D. Count duplicates
Answer: C
Explanation: A monotonic stack helps in comparing current elements efficiently to find next greater/smaller values.
Q4. [Definition] What does "Next Greater Element" mean for an element arr[i]?
A. Largest number after i
B. First number after i greater than arr[i]
C. Minimum number in array
D. All greater numbers
Answer: B
Explanation: NGE is the first element to the right of arr[i] that is greater than it.

Q5. [Definition] What is the time complexity of finding Next Greater Element using a stack?
A. $O(n^2)$
B. O(n log n)
C. O(n)
D. O(log n)
Answer: C
Explanation: Each element is pushed and popped once from the stack — resulting in linear time.
Q6. [Definition] Which data structure helps in solving Next Greater Element in O(n) time?
A. Queue
B. Min Heap
C. HashMap
D. Stack
Answer: D
Explanation: A stack (often monotonic) is used to solve NGE problems efficiently in O(n) time.
Q7. [Definition] In NGE problems, what should happen when current element is greater than stack top?
A. Pop the top and assign current as NGE
B. Skip current
C. Push top again
D. Reset the array
Answer: A
Explanation: If current > stack top, current is the NGE of top; we pop and update.
Q8. [Moderate] What will the Next Greater Element array be for [2, 1, 3]?
A. [3, 3, -1]
B. [1, 2, 3]
C. [3, 3, 3]
D. [3, -1, -1]
Answer: A
Explanation: 3 is the next greater for both 2 and 1; 3 has no greater element, so -1.
Q9. [Moderate] What does the following code do?
```
for (int i = n - 1; i >= 0; i--) {
    while (!stack.isEmpty() && arr[i] >= stack.peek()) {
        stack.pop();
    }
    result[i] = stack.isEmpty() ? -1 : stack.peek();
    stack.push(arr[i]);
}
```
A. Finds next smaller element to the right
B. Finds previous smaller element
C. Finds next greater element
D. Finds max element
Answer: C
Explanation: This is a standard reverse traversal approach for Next Greater Element using a stack.
Q10. [Moderate] Why do we iterate from right to left in NGE problems?
A. To sort the array
B. Because stack stores next elements
C. To compare with future elements

D. To reduce complexity
Answer: C
Explanation: To find the next greater element, we must look ahead — which is easier when iterating backward.
Q11. [Moderate] What is the result of NGE for array [4, 5, 2, 25]?
A. [5, 25, 25, -1]
B. [5, -1, 25, 25]
C. [25, 25, -1, -1]
D. [2, 5, 25, -1]
Answer: A
Explanation: For 4 → 5, for 5 → 25, for 2 → 25, for 25 → none.
Q12. [Moderate] Which condition keeps the stack in decreasing order?
A. While stack.top < current
B. While stack.top > current
C. While stack.top == current
D. While stack is not empty
Answer: A
Explanation: To maintain decreasing order, we pop while the current is greater.
Q13. [Moderate] What happens if we don't pop smaller elements from the stack in NGE logic?
A. Stack overflows
B. Wrong results — smaller elements block visibility of greater ones
C. Infinite loop
D. Slower performance only
Answer: B
Explanation: Popping smaller elements ensures the correct greater element is exposed.
Q14. [Moderate] What is stored in the stack during Next Greater Element logic?
A. Indices of elements
B. Sum of elements
C. Only max value
D. Sorted array
Answer: A (or values depending on variant)
Explanation: Usually, values or indices are stored depending on implementation; values directly for basic NGE.
Q15. [Moderate] For array [1, 3, 2, 4], what is the NGE array?
A. [3, 4, 4, -1]
B. [4, 3, 2, 1]
C. [2, 3, 4, -1]
D. [3, 2, 1, -1]
Answer: A
Explanation: 3 is next for 1, 4 is next for 3 and 2, no greater for 4.
Q1. [Definition] What is the principle of a queue?
A. LIFO (Last In First Out)
B. FIFO (First In First Out)
C. FILO (First In Last Out)
D. Random In Random Out
Answer: B
Explanation: Queue follows FIFO — the first element inserted is the first to be removed.

Q2. [Definition] What distinguishes a circular queue from a regular queue?
A. It supports deletion from front
B. Rear wraps around to front when full
C. It can be infinite
D. It uses a stack internally
Answer: B
Explanation: In a circular queue, rear wraps to the front if space is available, maximizing array use.
Q3. [Definition] Which condition determines queue overflow in a fixed-size circular queue (using array of size n)?
A. front == rear
B. (rear + 1) % n == front
C. front == -1
D. rear == 0
Answer: B
Explanation: When the next position of rear overlaps with front, the queue is full.
Q4. [Definition] What is a deque (double-ended queue)?
A. A queue that allows insertions only at front
B. A queue with a fixed size
C. A queue that allows insertion and deletion at both ends
D. A type of stack
Answer: C
Explanation: A deque supports insertions and deletions from both front and rear.
Q5. [Definition] In a normal queue, which operations are allowed?
A. pushFront, pushRear
B. enqueue, dequeue
C. popBack, popFront
D. insertAny, deleteAny
Answer: B
Explanation: Enqueue (insert at rear) and dequeue (remove from front) are standard queue operations.
Q6. [Definition] Which use case is best suited for a deque?
A. LRU Cache
B. Balanced parentheses
C. Binary search
D. Sorting
Answer: A
Explanation: Deques allow constant-time addition/removal from both ends — ideal for LRU caching.
Q7. [Definition] What is the time complexity of inserting or deleting from front or rear in a deque implemented using a linked list?
A. $O(1)$
B. $O(n)$
C. $O(\log n)$
D. $O(n^2)$
Answer: A
Explanation: Deque operations at both ends are constant time when implemented with linked lists.

Q8. [Moderate] What is the initial value of front and rear in an empty circular queue?
A. 0 and 0
B. 1 and 1
C. -1 and -1
D. 0 and -1
Answer: C
Explanation: Both are initialized to -1 to indicate the queue is empty.
Q9. [Moderate] What operation does this perform in a circular queue of size n?
rear = (rear + 1) % n;
A. Remove from rear
B. Reset rear to 0
C. Advance rear with wrap-around
D. Shift queue
Answer: C
Explanation: This moves rear one step forward, wrapping around if needed.
Q10. [Moderate] What condition checks if a circular queue is empty?
A. front == rear
B. front == -1
C. rear == -1
D. front == rear == 0
Answer: B
Explanation: In circular queues, front == -1 indicates emptiness.
Q11. [Moderate] What should be done after removing the last element in a circular queue?
A. front++
B. rear--
C. Set both front and rear to -1
D. Do nothing
Answer: C
Explanation: If the queue becomes empty, both pointers must be reset to -1.
Q12. [Moderate] In a deque, which function inserts an element at the front?
A. insertRear()
B. deleteFront()
C. pushFront()
D. addBack()
Answer: C
Explanation: pushFront() inserts elements at the front of a deque.
Q13. [Moderate] If rear == front, what does it indicate in a circular queue (after an operation)?
A. Queue is full
B. Queue is empty
C. One element present
D. Rear is ahead of front
Answer: C
Explanation: When both pointers are equal but not -1, only one element is in the queue.

Q14. [Moderate] What output does this produce?
Deque<Integer> dq = new ArrayDeque<>();
dq.offerLast(10);

dq.offerFirst(20);
System.out.println(dq.peekFirst());
A. 10
B. 20
C. 30
D. null
Answer: B
Explanation: offerFirst(20) places 20 at the front. The first element is 20.
Q15. [Moderate] Which of the following is not a valid deque operation?
A. offerFirst()
B. pollLast()
C. peekMiddle()
D. offerLast()
Answer: C
Explanation: Deques support operations at both ends, but there is no standard method like peekMiddle().
Q1. [Definition] What is the goal of the Sliding Window Maximum problem?
A. Find average of window elements
B. Find smallest element in array
C. Find the maximum value in each window of size k
D. Remove duplicates
Answer: C
Explanation: The task is to find the max value for each subarray (window) of size k.
Q2. [Definition] Which data structure is optimal for solving Sliding Window Maximum in O(n)?
A. Stack
B. Queue
C. Deque
D. HashSet
Answer: C
Explanation: A **monotonic deque** helps track max elements efficiently in O(n) time.
Q3. [Definition] In a monotonic deque used for Sliding Window Maximum, which elements are removed from the back?
A. Smaller than current
B. Greater than current
C. Equal to current
D. All of them
Answer: A
Explanation: Smaller elements are removed as they can't be max for any upcoming window.
Q4. [Definition] What condition is checked to remove elements from the front of the deque?
A. Value is even
B. Index is out of current window
C. Value is less than max
D. Index is in window
Answer: B
Explanation: If the index is outside the current window, it's removed from the front.
Q5. [Definition] What's the time complexity of solving Sliding Window Maximum using deque?
A. $O(n^2)$

B. O(n log n)
C. O(n)
D. O(k log n)
Answer: C
Explanation: Each element is added and removed from the deque at most once — total O(n).
Q6. [Definition] What does it mean to implement a stack using a queue?
A. FIFO behavior
B. Insert elements normally, reverse order on pop
C. LIFO behavior using FIFO structure
D. Stack becomes queue
Answer: C
Explanation: A stack can be implemented with queue(s) by simulating LIFO using FIFO operations.
Q7. [Definition] What is the key operation in queue-based stack implementation to simulate LIFO?
A. Push to back
B. Enqueue to front
C. Rotate the queue after each insert
D. Reverse array
Answer: C
Explanation: After pushing, rotate the queue so that the newest element moves to the front.
Q8. [Moderate] What will be the output of Sliding Window Maximum for array [1,3,-1,-3,5,3,6,7] with k = 3?
A. [3,3,5,5,6,7]
B. [1,1,1,1,1,1]
C. [3,5,6,7,7,7]
D. [3,3,3,3,3,3]
Answer: A
Explanation: The max of each window is: [3,3,5,5,6,7].
Q9. [Moderate] What will this code do (Stack using 2 Queues)?
```
void push(int x) {
   q2.add(x);
   while (!q1.isEmpty()) {
      q2.add(q1.remove());
   }
   Queue<Integer> temp = q1;
   q1 = q2;
   q2 = temp;
}
```
A. Implements pop operation
B. Simulates LIFO with two queues
C. Stack overflow
D. Rotates a queue
Answer: B
Explanation: The newest element is inserted at the front by emptying q1 into q2 after pushing.
Q10. [Moderate] Which function retrieves the current window's max efficiently in deque approach?
A. peekLast()

B. getMax() from front
C. getMin()
D. Binary search
Answer: B
Explanation: The front of the deque always contains the current window's max.
Q11. [Moderate] Which of the following is NOT a valid step in implementing Queue using Stack (2 Stacks)?
A. Push to input stack
B. Pop from output stack
C. Push to both stacks
D. Transfer from input to output stack if output is empty
Answer: C
Explanation: There's no need to push to both — input receives elements, and output handles reverse order.
Q12. [Moderate] In queue using 2 stacks, why do we transfer elements from one to another only when needed?
A. To improve speed
B. To maintain order
C. To reduce space
D. To break LIFO
Answer: B
Explanation: Elements are reversed once to maintain the original insertion order (FIFO).
Q13. [Moderate] In a Sliding Window Max, what happens if i - k + 1 > deque.peekFirst()?
A. No action
B. We discard the front index
C. We reset the window
D. We add i to deque
Answer: B
Explanation: It means the index at the front is out of window and must be removed.
Q14. [Moderate] What is the space complexity of Sliding Window Maximum using deque?
A. O(k)
B. O(n)
C. O(1)
D. $O(n^2)$
Answer: A
Explanation: The deque at most holds k indices (one window's worth).

Q15. [Moderate] If a stack is implemented using a single queue, how do we simulate pop()?
A. Remove from front
B. Rotate all elements except last, then remove front
C. Sort and remove max
D. Use peek only
Answer: B
Explanation: Rotate elements till the last is at the front, then remove it to simulate LIFO.
Q1. [Definition] A pile of plates is stacked in a canteen. The last plate placed is removed first. What data structure does this represent?
A. Queue

B. Stack
C. Tree
D. Graph
Answer: B
Explanation: Like plates, a stack follows LIFO (Last In First Out) behavior.
Q2. [Definition] A browser keeps track of visited web pages. When the user presses "Back", it goes to the last page. What data structure is used here?
A. Queue
B. Stack
C. Array
D. Heap
Answer: B
Explanation: Browser back/forward uses a stack to remember the previous page.
Q3. [Definition] A person opens boxes within boxes. To get to the first box, all others must be closed in reverse order. Which structure is this?
A. Queue
B. Linked List
C. Stack
D. HashMap
Answer: C
Explanation: Unboxing nested containers simulates LIFO behavior of a stack.
Q4. [Definition] A person matches socks from a laundry basket. Each sock is picked and matched in reverse order. What concept does this illustrate?
A. Hashing
B. Queue
C. Valid Parentheses using Stack
D. Sorting
Answer: C
Explanation: Matching opening and closing items, like socks or brackets, resembles stack-based validation.
Q5. [Definition] A magician uses a trick where every card has a matching pair. He pushes each card into a pile and pops it when its pair is found. What problem does this relate to?
A. Sorting
B. Palindrome
C. Parentheses matching
D. Maximum subarray
Answer: C
Explanation: Like brackets, matching cards represent the valid parentheses concept using a stack.
Q6. [Definition] A teacher asks students to submit homework. The last to submit is graded first. What structure models this?
A. Stack
B. Queue
C. Array
D. Set
Answer: A
Explanation: The last-in assignment is picked first — classic stack behavior.

Q7. [Definition] A diver descends into the sea layer by layer. To return, they must ascend layer by layer in reverse. Which concept fits?
A. Queue
B. Tree
C. Stack
D. DFS
Answer: C
Explanation: Recursive descent and reverse ascent reflect stack-like behavior.
Q8. [Definition] A student writes on a blackboard and uses undo to erase the most recent word. What data structure does this resemble?
A. Queue
B. Heap
C. Stack
D. Graph
Answer: C
Explanation: Undo operations are stored in a stack to remove the latest change.
Q9. [Definition] A person climbs a mountain and notes elevation at each point. To find when they last climbed higher, they use a special notebook. This models which algorithm?
A. BFS
B. Next Greater Element using Stack
C. Binary Search
D. Backtracking
Answer: B
Explanation: This is similar to finding the next greater height using a monotonic stack.
Q10. [Definition] A soldier keeps armor pieces stacked. The weakest is always placed on top. At any time, he can retrieve the weakest piece. Which data structure supports this?
A. Priority Queue
B. HashMap
C. Min Stack
D. Graph
Answer: C
Explanation: Min Stack helps retrieve the minimum element in constant time while supporting push/pop.
Q11. [Definition] A security guard records maximum temperatures over rolling 3-day windows. Each time, he updates the hottest day in that period. Which technique does he use?
A. Prefix sum
B. Deque for Sliding Window Maximum
C. Binary Heap
D. Brute force
Answer: B
Explanation: This is exactly what the sliding window maximum problem does using a monotonic deque.
Q12. [Definition] A robot picks objects and keeps them in a pile. To access the bottom item, it has to remove all items above it. Which access order is used here?
A. Random Access
B. FIFO
C. LIFO

D. DFS
Answer: C
Explanation: Accessing the last inserted item first is LIFO.
Q13. [Definition] A restaurant has an automatic tray system. Trays go into a stack and are dispensed in reverse order of insertion. Which structure matches this behavior?
A. Queue
B. Deque
C. Stack
D. Tree
Answer: C
Explanation: Stacked trays are dispensed in LIFO order.
Q14. [Definition] A line of people each has a height. For each person, you want to know the next taller person. What technique should be used?
A. HashSet
B. Sliding Window
C. Monotonic Stack for NGE
D. Stack with sorting
Answer: C
Explanation: This is the classic Next Greater Element problem using a monotonic decreasing stack.
Q15. [Definition] A person pushes items into a stack using a queue. After each push, the newest item moves to the front. What concept is this?
A. Queue using stack
B. Stack using queue
C. Circular queue
D. Min heap
Answer: B
Explanation: Rotating after each push simulates LIFO behavior using a queue.
Q16. [Definition] A warehouse tracks the smallest box placed at any moment. When removing boxes, it always knows the current smallest. What structure helps?
A. Queue
B. Min Stack
C. Heap
D. BST
Answer: B
Explanation: Min Stack supports push/pop with getMin in constant time.
Q17. [Definition] A man opens doors in a house. He can only open one door at a time and must close the current door before opening the previous. What concept does this reflect?
A. Queue
B. Stack
C. DFS
D. Priority Queue
Answer: B
Explanation: The behavior matches how a stack unwinds — close last opened, then previous.
Q18. [Definition] A mail sorter pushes letters into a bag. The topmost letter is always removed first. The bag mimics what structure?
A. Array
B. Queue

C. Stack
D. HashSet
Answer: C
Explanation: This mimics stack-like LIFO behavior.
Q19. [Definition] A toy sorter must quickly remove the largest toy within the last 5 toys inserted. Which approach helps here?
A. Max Stack
B. Deque with sliding window
C. Priority Queue
D. HashMap
Answer: B
Explanation: Sliding window maximum uses a deque to track max within recent elements.
Q20. [Definition] A magician stores tricks in a book and removes the most recent one to perform. He also remembers which one was the easiest. Which data structure supports both?
A. Queue
B. Stack
C. Min Stack
D. TreeMap
Answer: C
Explanation: Min Stack supports constant-time retrieval of the minimum and normal stack operations.