1. Implementation of Stack Using Array

1. **Which property best describes a stack?**
   a) First In First Out (FIFO)
   b) Last In First Out (LIFO)
   c) First In Last Out (FILO)
   d) Last In Last Out (LILO)
   Answer: b) Last In First Out (LIFO)
2. **Which operation is used to add an element to a stack?**
   a) pop
   b) push
   c) enqueue
   d) insert
   Answer: b) push
3. **In an array-based stack implementation, what variable typically keeps track of the top element?**
   a) front index
   b) rear index
   c) top index
   d) bottom index
   Answer: c) top index
4. **What is the time complexity for both push and pop operations in an array-based stack?**
   a) O(n)
   b) O(log n)
   c) O(1)
   d) $O(n^2)$
   Answer: c) O(1)
5. **When does a stack overflow occur in an array-based stack?**
   a) When top equals –1
   b) When top equals MAX_SIZE – 1
   c) When top is less than 0
   d) When top equals MAX_SIZE
   Answer: b) When top equals MAX_SIZE – 1
6. **What happens if you try to pop an element from an empty stack?**
   a) It returns the bottom element
   b) It causes a stack underflow error
   c) It wraps around to the last element
   d) It does nothing
   Answer: b) It causes a stack underflow error
7. **Which application is most commonly implemented using a stack?**
   a) Sorting an array
   b) Infix to postfix expression conversion
   c) Binary search
   d) Queue simulation
   Answer: b) Infix to postfix expression conversion

8. **If an array-based stack of size 10 already contains 10 elements, what will happen when you try to push an 11th element?**
   a) The element is added at the beginning
   b) The element replaces the last element
   c) A stack overflow error occurs
   d) The stack automatically resizes
   Answer: c) A stack overflow error occurs
9. **In an array-based stack, which statement is true about accessing elements?**
   a) You can access any element randomly
   b) Only the top element is directly accessible
   c) Both the first and last elements are equally accessible
   d) Elements are accessed via a linked list
   Answer: b) Only the top element is directly accessible
10. **Which operation allows you to view the top element without removing it?**
    a) pop
    b) peek
    c) push
    d) top
    Answer: b) peek

2. Implementation of Queue Using Array

1. **Which property best describes a queue?**
   a) Last In First Out (LIFO)
   b) First In First Out (FIFO)
   c) First In Last Out (FILO)
   d) Last In Last Out (LILO)
   Answer: b) First In First Out (FIFO)
2. **Which operation is used to add an element to a queue?**
   a) pop
   b) push
   c) enqueue
   d) dequeue
   Answer: c) enqueue
3. **Which operation is used to remove an element from a queue?**
   a) pop
   b) enqueue
   c) push
   d) dequeue
   Answer: d) dequeue
4. **In an array-based queue implementation, which pointers are typically maintained?**
   a) Only a front pointer
   b) Only a rear pointer
   c) Both front and rear pointers

d) Neither front nor rear pointers
Answer: c) Both front and rear pointers

5. **What is the time complexity for enqueue and dequeue operations in a circular queue?**
   a) O(1)
   b) O(n)
   c) O(log n)
   d) $O(n^2)$
   Answer: a) O(1)
6. **What problem does a circular queue solve in an array-based implementation?**
   a) Stack overflow
   b) Wasted space due to non-reuse of freed positions
   c) Memory leak
   d) Underflow errors
   Answer: b) Wasted space due to non-reuse of freed positions
7. **In a simple linear array implementation of a queue, what issue might occur when rear reaches the end of the array while there is free space at the front?**
   a) Queue underflow
   b) Automatic resizing
   c) Queue overflow even though space is available
   d) Elements are automatically shifted
   Answer: c) Queue overflow even though space is available
8. **In many array-based queue implementations, how is an empty queue commonly detected?**
   a) When front equals rear
   b) When front equals –1
   c) When rear equals –1
   d) When front equals MAX_SIZE
   Answer: b) When front equals –1
   (Note: Some implementations use front == rear; the answer depends on the chosen initialization.)
9. **In a circular queue, how do you calculate the new position of rear after an insertion?**
   a) rear = rear + 1
   b) rear = (rear + 1) % MAX_SIZE
   c) rear = rear – 1
   d) rear = front + 1
   Answer: b) rear = (rear + 1) % MAX_SIZE
10. **What is a major drawback of using a simple linear array for queue implementation?**
    a) It supports dynamic resizing automatically
    b) It may waste space after several dequeue operations
    c) It always uses O(1) space
    d) It automatically wraps around when full
    Answer: b) It may waste space after several dequeue operations

3. Infix to Postfix Conversion

1. **Which of the following is an example of an infix expression?**
   a) AB+
   b) A+B
   c) +AB
   d) AB+*
   Answer: b) A+B
2. **In a postfix expression, where is the operator placed relative to its operands?**
   a) Before the operands
   b) After the operands
   c) Between the operands
   d) It is not used
   Answer: b) After the operands
3. **Which data structure is primarily used for converting infix expressions to postfix?**
   a) Queue
   b) Stack
   c) Linked List
   d) Tree
   Answer: b) Stack
4. **What is the primary purpose of the stack in infix to postfix conversion?**
   a) To store operands
   b) To store operators and manage precedence
   c) To store the final postfix expression
   d) To reverse the input expression
   Answer: b) To store operators and manage precedence
5. **When scanning an infix expression, what should you do when you encounter an operand?**
   a) Push it onto the stack
   b) Add it directly to the output
   c) Ignore it
   d) Replace it with an operator
   Answer: b) Add it directly to the output
6. **How are parentheses handled during the conversion from infix to postfix?**
   a) They are added directly to the output
   b) They are pushed on the stack and popped when a matching parenthesis is found
   c) They are ignored
   d) They are converted into operators
   Answer: b) They are pushed on the stack and popped when a matching parenthesis is found
7. **Which operator typically has a higher precedence?**
   a) +
   b) -
   c) *

d) All operators have equal precedence
Answer: c) *

8. **After completely scanning an infix expression, what is the final step in the conversion process?**
   a) Discard the stack
   b) Output the remaining operands
   c) Pop all remaining operators from the stack and add them to the output
   d) Reverse the output
   Answer: c) Pop all remaining operators from the stack and add them to the output
9. **What is the postfix form of the infix expression "A+B*C"?**
   a) AB+C*
   b) ABC*+
   c) AC*B*+
   d) A+BC
   Answer: b) ABC*+
10. **When an incoming operator has lower precedence than the operator on the stack, what is the correct action?**
    a) Push the incoming operator onto the stack
    b) Pop the operator from the stack and add it to the output
    c) Discard the incoming operator
    d) Swap the operators
    Answer: b) Pop the operator from the stack and add it to the output

4. Doubly Linked List Implementation Using Structure

1. **What is a doubly linked list?**
   a) A list with exactly two nodes
   b) A list where each node has pointers to both its next and previous nodes
   c) A list implemented using an array
   d) A list where nodes only point to the next node
   Answer: b) A list where each node has pointers to both its next and previous nodes
2. **What are the two pointers typically called in a doubly linked list node?**
   a) front and rear
   b) head and tail
   c) next and prev
   d) top and bottom
   Answer: c) next and prev
3. **Which advantage does a doubly linked list have over a singly linked list?**
   a) Easier random access
   b) Ability to traverse both forward and backward
   c) Uses less memory
   d) Faster search operations
   Answer: b) Ability to traverse both forward and backward

4. **What is the time complexity for inserting a node at the beginning of a doubly linked list?**
   a) O(n)
   b) O(log n)
   c) O(1)
   d) $O(n^2)$
   Answer: c) O(1)
5. **When deleting a node from a doubly linked list, what extra pointer update is required compared to a singly linked list?**
   a) Update only the next pointer of the previous node
   b) Update only the prev pointer of the next node
   c) Update both the next pointer of the previous node and the prev pointer of the next node
   d) No additional update is needed
   Answer: c) Update both the next pointer of the previous node and the prev pointer of the next node
6. **What happens if you attempt to traverse backward from the head of a doubly linked list?**
   a) You reach the tail
   b) You encounter a null pointer
   c) The list loops indefinitely
   d) The program crashes
   Answer: b) You encounter a null pointer
7. **How do you typically identify the end of a doubly linked list?**
   a) When the next pointer is NULL
   b) When the prev pointer is NULL
   c) When both pointers are non-NULL
   d) When the node's data is NULL
   Answer: a) When the next pointer is NULL
8. **Which operation is generally more efficient in a doubly linked list compared to an array when inserting or deleting in the middle?**
   a) Insertion only
   b) Deletion only
   c) Both insertion and deletion
   d) Neither operation
   Answer: c) Both insertion and deletion
9. **What is the primary disadvantage of a doubly linked list compared to a singly linked list?**
   a) Slower forward traversal
   b) Higher memory consumption due to an extra pointer
   c) Inability to traverse backward
   d) More complex searching
   Answer: b) Higher memory consumption due to an extra pointer
10. **In C, what is most commonly used to define a node in a doubly linked list?**
    a) A class
    b) A structure
    c) An array

d) A pointer only
Answer: b) A structure

5. Singly Linked List Using Structure

1. **What is a singly linked list?**
   a) A list where each node points to both next and previous nodes
   b) A list where each node points only to the next node
   c) A list implemented using an array
   d) A doubly linked list
   Answer: b) A list where each node points only to the next node
2. **What is the first node of a singly linked list called?**
   a) Head
   b) Tail
   c) Front
   d) Root
   Answer: a) Head
3. **What is the time complexity for inserting a node at the beginning of a singly linked list?**
   a) O(n)
   b) O(log n)
   c) O(1)
   d) $O(n^2)$
   Answer: c) O(1)
4. **What does each node in a singly linked list typically contain?**
   a) Data and both next and previous pointers
   b) Data and a next pointer
   c) Only data
   d) Data and an index
   Answer: b) Data and a next pointer
5. **How do you traverse a singly linked list?**
   a) Using an index-based for loop
   b) By following the next pointer from the head until NULL
   c) By accessing elements randomly
   d) Using binary search
   Answer: b) By following the next pointer from the head until NULL
6. **What is the time complexity of searching for an element in a singly linked list?**
   a) O(1)
   b) O(log n)
   c) O(n)
   d) $O(n^2)$
   Answer: c) O(n)
7. **Which operation is particularly efficient in a singly linked list compared to an array?**

a) Random access
b) Insertion at the beginning
c) Sorting
d) Binary search
Answer: b) Insertion at the beginning

8. **What does a NULL pointer typically signify in a linked list?**
   a) The start of the list
   b) The end of the list
   c) A duplicate node
   d) An error in the list
   Answer: b) The end of the list
9. **What can cause a memory leak in a singly linked list implementation?**
   a) Not updating the head pointer
   b) Not freeing memory when a node is deleted
   c) Repeated traversal of the list
   d) Using recursion for traversal
   Answer: b) Not freeing memory when a node is deleted
10. **Which algorithm is commonly used to detect a loop in a singly linked list?**
    a) Using a counter variable
    b) Floyd's cycle-finding algorithm
    c) Binary search
    d) Merge sort
    Answer: b) Floyd's cycle-finding algorithm

6. Recursion (Fibonacci, Factorial, Tower of Hanoi)

1. **What is recursion?**
   a) A function calling itself
   b) A loop that repeats
   c) A function that calls another function
   d) A function that returns void
   Answer: a) A function calling itself
2. **What is the purpose of the base case in a recursive function?**
   a) To continue the recursion indefinitely
   b) To stop the recursion
   c) To call another function
   d) To initialize the recursion
   Answer: b) To stop the recursion
3. **Which of the following correctly defines the factorial of n (n!) recursively?**
   a) $n! = n \times (n - 1)!$
   b) $n! = n + (n - 1)!$
   c) $n! = (n - 1)! \div n$
   d) $n! = n - (n - 1)!$
   Answer: a) $n! = n \times (n - 1)!$

4. **What is the main drawback of the simple recursive implementation of the Fibonacci series?**
   a) High memory usage
   b) Redundant calculations leading to exponential time complexity
   c) It cannot handle large numbers
   d) It requires iterative loops
   Answer: b) Redundant calculations leading to exponential time complexity
5. **For n disks in the Tower of Hanoi problem, what is the minimum number of moves required?**
   a) $2^n - 1$
   b) n^2
   c) n!
   d) 2n – 1
   Answer: a) $2^n - 1$
6. **One advantage of using recursion is that it can simplify code for problems that involve:**
   a) Complex loops
   b) Repetitive subproblems with similar structure
   c) Direct random access
   d) Low-level memory management
   Answer: b) Repetitive subproblems with similar structure
7. **What is tail recursion?**
   a) A recursive function where the recursive call is the last statement in the function
   b) A recursion with no base case
   c) A recursive function with multiple recursive calls
   d) A recursion that calls another function before returning
   Answer: a) A recursive function where the recursive call is the last statement in the function
8. **Which technique can optimize a recursive Fibonacci function by storing intermediate results?**
   a) Iteration
   b) Memoization
   c) Binary search
   d) Loop unrolling
   Answer: b) Memoization
9. **What is the space complexity of a recursive function in terms of its call stack usage (assuming n recursive calls)?**
   a) O(1)
   b) O(n)
   c) O(log n)
   d) $O(n^2)$
   Answer: b) O(n)
10. **In the context of recursion, what does "stack overflow" refer to?**
    a) Running out of heap memory
    b) Running out of call stack space due to too many recursive calls
    c) An error in an array-based stack implementation

d) Excessive memory allocated for dynamic arrays
Answer: b) Running out of call stack space due to too many recursive calls

7. Sorting (Bubble, Insertion, Selection)

1. **Which sorting algorithm repeatedly swaps adjacent elements if they are in the wrong order?**
   a) Insertion Sort
   b) Bubble Sort
   c) Selection Sort
   d) Merge Sort
   Answer: b) Bubble Sort
2. **What is the best-case time complexity of Bubble Sort (with an optimized version using a flag)?**
   a) O(n)
   b) O(n log n)
   c) $O(n^2)$
   d) O(1)
   Answer: a) O(n)
3. **Which sorting algorithm builds the sorted array one element at a time by comparing each new element to the already sorted part?**
   a) Bubble Sort
   b) Selection Sort
   c) Insertion Sort
   d) Quick Sort
   Answer: c) Insertion Sort
4. **What is the primary idea behind Selection Sort?**
   a) Swapping adjacent elements repeatedly
   b) Inserting each element into its correct position
   c) Finding the minimum element from the unsorted part and swapping it with the first unsorted element
   d) Dividing the array and merging sorted halves
   Answer: c) Finding the minimum element from the unsorted part and swapping it with the first unsorted element
5. **What is the average-case time complexity of Selection Sort?**
   a) O(n)
   b) O(n log n)
   c) $O(n^2)$
   d) O(log n)
   Answer: c) $O(n^2)$
6. **Insertion Sort is particularly efficient for which type of input?**
   a) Large randomly ordered arrays
   b) Nearly sorted arrays
   c) Arrays with all identical elements

d) Arrays sorted in reverse order
Answer: b) Nearly sorted arrays

7. **Which of the following sorting algorithms is considered stable (i.e., does not change the relative order of equal elements) in its standard form?**
   a) Selection Sort
   b) Insertion Sort
   c) Both Selection Sort and Insertion Sort
   d) Neither Selection Sort nor Insertion Sort
   Answer: b) Insertion Sort
   (Note: Bubble Sort is also stable, but here we focus on the three mentioned.)
8. **How many swaps does Selection Sort perform in the worst case?**
   a) O(n) swaps (exactly n–1 swaps)
   b) O(n log n) swaps
   c) $O(n^2)$ swaps
   d) O(1) swap
   Answer: a) O(n) swaps (exactly n–1 swaps)
9. **Which of the following sorting algorithms is in-place?**
   a) Bubble Sort
   b) Insertion Sort
   c) Selection Sort
   d) All of the above
   Answer: d) All of the above
10. **What is a common drawback of Bubble Sort compared to more efficient algorithms like Quick Sort?**
    a) It is unstable
    b) It has an average and worst-case time complexity of $O(n^2)$
    c) It requires additional memory
    d) It cannot handle duplicate values
    Answer: b) It has an average and worst-case time complexity of $O(n^2)$