Session 1

🟢 EASY LEVEL (1–20)

1. What is the first step in problem solving?

A. Coding
B. Testing
C. Defining the problem
D. Documentation

Answer: ✅ C
Explanation: Clearly defining the problem avoids wrong solutions.

---

2. Computational thinking mainly focuses on:

A. Writing programs
B. Breaking problems into smaller parts
C. Using computers only
D. Learning syntax

Answer: ✅ B

---

3. Which of the following best describes a “problem”?

A. A program with errors
B. A situation requiring a solution
C. A flowchart
D. A test case

Answer: ✅ B

---

4. Identifying inputs and outputs is part of:

A. Coding
B. Testing
C. Problem definition
D. Debugging

Answer: ✅ C

---

5. Which tool visually represents logic using symbols?

A. Pseudocode
B. Algorithm
C. Flowchart
D. Source code

Answer: ✅ C

---

6. Pseudocode is:

A. Executable code
B. Machine language
C. Informal description of algorithm
D. Compiler output

Answer: ✅ C

---

7. Which of these is NOT a problem-solving technique?

A. Decomposition
B. Abstraction
C. Compilation
D. Pattern recognition

Answer: ✅ C

---

8. Real-world problem solving mainly requires:

A. Syntax knowledge
B. Logical thinking
C. Compiler
D. IDE

Answer: ✅ B

---

9. Flowchart decision symbol is represented by:

A. Rectangle
B. Circle
C. Diamond
D. Oval

Answer: ✅ C

---

10. What does decomposition mean?

A. Combining solutions
B. Breaking problem into smaller parts
C. Writing code
D. Testing software

Answer: ✅ B

---

11. Which step comes after problem definition?

A. Debugging
B. Identifying solution approach
C. Execution
D. Deployment

Answer: ✅ B

---

12. Which symbol represents start/end in a flowchart?

A. Rectangle
B. Oval
C. Diamond
D. Arrow

Answer: ✅ B

---

13. Pseudocode improves:

A. Compiler speed
B. Program execution
C. Human understanding
D. Memory usage

Answer: ✅ C

---

14. Selecting best solution depends on:

A. Random choice
B. Selection criteria
C. Syntax rules
D. Language used

Answer: ✅ B

---

15. Which is NOT an advantage of flowcharts?

A. Easy understanding
B. Clear logic flow
C. Faster execution
D. Visual clarity

Answer: ✅ C

---

16. Problem solving is:

A. Only coding
B. Trial and error
C. Systematic approach
D. Guessing

Answer: ✅ C

---

17. Identifying constraints is part of:

A. Coding
B. Problem analysis
C. Testing
D. Maintenance

Answer: ✅ B

---

18. Computational thinking helps mainly in:

A. Writing syntax
B. Designing solutions

C. Compiling code
D. Printing output

Answer: ✅ B

---

19. Flowchart arrows indicate:

A. Variables
B. Direction of control
C. Decisions
D. Data

Answer: ✅ B

---

20. Mini projects mainly help in:

A. Memorization
B. Practical understanding
C. Syntax learning
D. Debugging only

Answer: ✅ B

🟡 MEDIUM LEVEL (21–40)

21. Which is the correct order of problem-solving steps?

A. Code → Test → Define
B. Define → Analyze → Design → Implement
C. Analyze → Define → Code
D. Test → Code → Define

Answer: ✅ B

---

22. Identify the correct pseudocode element:

```
IF marks > 40
  PRINT "Pass"
ELSE
  PRINT "Fail"
```

This represents:
A. Sequence
B. Selection
C. Iteration
D. Recursion

Answer: ✅ B

---

23. Which flowchart symbol is used for input/output?

A. Oval
B. Rectangle
C. Parallelogram
D. Diamond

Answer: ✅ C

---

24. Abstraction means:

A. Ignoring irrelevant details
B. Writing full code
C. Debugging
D. Testing

Answer: ✅ A

---

25. Real-world example of decomposition:

A. Writing one large function
B. Dividing project into modules
C. Debugging code
D. Running tests

Answer: ✅ B

---

26. Which is NOT a selection criterion?

A. Time complexity
B. Space usage
C. Readability
D. Keyboard type

Answer: ✅ D

---

27. In problem solving, constraints refer to:

A. Output
B. Limitations
C. Flowchart symbols
D. Variables

Answer: ✅ B

---

28. Which technique finds similarities in problems?

A. Decomposition
B. Abstraction
C. Pattern recognition
D. Algorithm

Answer: ✅ C

---

29. What comes after selecting the solution?

A. Documentation
B. Implementation
C. Testing
D. Debugging

Answer: ✅ B

---

30. Flowchart mainly helps in:

A. Execution
B. Debugging only
C. Understanding logic
D. Compilation

Answer: ✅ C

---

31. Pseudocode should be:

A. Strictly syntactic
B. Language dependent

C. Simple English-like
D. Machine-readable

Answer: ✅ C

---

32. Which step includes writing algorithm?

A. Problem identification
B. Solution design
C. Testing
D. Deployment

Answer: ✅ B

---

33. Team-based problem solving improves:

A. Syntax
B. Logical discussion
C. Compiler errors
D. Hardware

Answer: ✅ B

---

34. Which is a real-world problem?

A. Printing numbers
B. Online food delivery optimization
C. Hello World
D. Variable declaration

Answer: ✅ B

---

35. Flowchart decision symbol has:

A. One exit
B. Two exits
C. No exit
D. Multiple inputs

Answer: ✅ B

---

36. Documentation is important because:

A. Compiler needs it
B. Helps future understanding
C. Increases speed
D. Reduces memory

Answer: ✅ B

---

37. Which is an example of abstraction?

A. Writing all details
B. Showing only main steps
C. Debugging
D. Running code

Answer: ✅ B

---

38. Problem identification includes:

A. Coding
B. Understanding requirements
C. Testing
D. Deployment

Answer: ✅ B

---

39. Mini project selection depends on:

A. Random choice
B. Interest and feasibility
C. Syntax
D. Compiler

Answer: ✅ B

---

40. Which improves computational thinking most?

A. Memorization
B. Practice with real problems
C. Reading syntax
D. Using IDE

Answer: ✅ B

🔴 HARD LEVEL (41–60)

41. Which pseudocode correctly finds maximum of two numbers?

A.

```
IF a > b
PRINT b
```

B.

```
IF a > b
PRINT a
ELSE
PRINT b
```

C.

```
PRINT a > b
```

D.

```
IF a < b PRINT a
```

Answer: ✅ B

---

42. Flowchart start → input → decision → output → end represents:

A. Sequence only
B. Selection structure
C. Loop
D. Recursion

Answer: ✅ B

---

43. A poorly defined problem leads to:

A. Optimal solution
B. Wrong solution
C. Faster execution
D. Better documentation

Answer: ✅ B

---

44. Which is the best solution selection criterion?

A. Fastest to code
B. Least logical
C. Efficient and feasible
D. Longest code

Answer: ✅ C

---

45. Which step converts idea into working system?

A. Analysis
B. Design
C. Implementation
D. Documentation

Answer: ✅ C

---

46. Consider flowchart with repeated decision back to input. It represents:

A. Sequence
B. Selection
C. Iteration
D. Recursion

Answer: ✅ C

---

47. Which problem-solving technique hides internal complexity?

A. Decomposition
B. Abstraction
C. Iteration
D. Testing

Answer: ✅ B

---

48. Choosing wrong algorithm affects:

A. Syntax
B. Performance
C. Compiler
D. Editor

Answer: ✅ B

---

49. A solution that works for only one input is:

A. General solution
B. Partial solution
C. Optimal solution
D. Complete solution

Answer: ✅ B

---

50. Computational thinking is language:

A. Dependent
B. Independent
C. Compiler-based
D. Hardware-based

Answer: ✅ B

---

51. Which step ensures solution correctness?

A. Coding
B. Testing
C. Design
D. Analysis

Answer: ✅ B

---

52. Real-world problems usually have:

A. Single solution
B. No constraints

C. Multiple constraints
D. No input

Answer: ✅ C

---

53. Best mini project idea should be:

A. Very complex
B. Copy-paste
C. Practical and manageable
D. Syntax heavy

Answer: ✅ C

---

54. Pseudocode is preferred before coding because:

A. Compiler needs it
B. Reduces logical errors
C. Increases speed
D. Reduces memory

Answer: ✅ B

---

55. Which is NOT part of documentation?

A. Problem statement
B. Algorithm
C. Code comments only
D. Flowchart

Answer: ✅ C

---

56. Flowchart helps in debugging by:

A. Running code
B. Showing logical flow
C. Printing output
D. Compiling

Answer: ✅ B

---

57. Which improves team-based analysis?

A. Individual coding
B. Group discussion
C. Random ideas
D. Copying

Answer: ✅ B

---

58. Selecting inefficient solution mainly impacts:

A. Output correctness
B. Performance
C. Syntax
D. Documentation

Answer: ✅ B

---

59. Computational thinking applies to:

A. Only programming
B. Only math
C. Any problem domain
D. Only computers

Answer: ✅ C

---

60. Final goal of problem solving is:

A. Writing code
B. Passing exam
C. Correct and efficient solution
D. Flowchart

Answer: ✅ C

Session 2 & 3

🟢 EASY LEVEL (1–50)

1. What is an algorithm?

A. Programming language
B. Finite set of steps to solve a problem
C. Hardware component
D. Compiler

Answer: ✅ B

---

2. Which property must an algorithm have?

A. Infinite steps
B. Ambiguity
C. Finiteness
D. Hardware dependency

Answer: ✅ C

---

3. Big-O notation represents:

A. Exact execution time
B. Worst-case time complexity
C. Best-case complexity
D. Memory address

Answer: ✅ B

---

4. Which data structure follows LIFO?

A. Queue
B. Stack
C. Array
D. Linked list

Answer: ✅ B

---

5. Which data structure follows FIFO?

A. Stack
B. Tree
C. Queue
D. Graph

Answer: ✅ C

---

6. Push operation is related to:

A. Queue
B. Stack
C. Array
D. Tree

Answer: ✅ B

---

7. Pop operation removes element from:

A. Bottom of stack
B. Middle of stack
C. Top of stack
D. Any position

Answer: ✅ C

---

8. Which is NOT a basic data structure?

A. Array
B. Stack
C. Queue
D. Compiler

Answer: ✅ D

---

9. Abstract Data Type (ADT) focuses on:

A. Implementation
B. Syntax
C. What operations are performed
D. Memory layout

Answer: ✅ C

---

10. Which is an example of ADT?

A. Stack
B. int array
C. Pointer
D. Variable

Answer: ✅ A

---

11. Time complexity of accessing array element by index is:

A. O(n)
B. O(log n)
C. O(1)
D. O(n²)

Answer: ✅ C

---

12. Which operation is NOT possible in stack?

A. push
B. pop
C. peek
D. insert at middle

Answer: ✅ D

---

13. Queue insertion is called:

A. Push
B. Pop
C. Enqueue
D. Dequeue

Answer: ✅ C

---

14. Queue deletion is called:

A. Push
B. Enqueue
C. Dequeue
D. Pop

Answer: ✅ C

---

15. Circular queue is used to:

A. Waste memory
B. Avoid memory wastage
C. Increase complexity
D. Slow operations

Answer: ✅ B

---

16. Which symbol represents algorithm logic visually?

A. Code
B. Pseudocode
C. Flowchart
D. Compiler

Answer: ✅ C

---

17. Stack overflow occurs when:

A. Stack is empty
B. Stack is full and push is done
C. Stack has one element
D. Stack grows dynamically

Answer: ✅ B

---

18. Stack underflow occurs when:

A. Stack is full
B. Pop on empty stack
C. Push on full stack
D. Peek operation

Answer: ✅ B

---

19. Which operation gives top element without removing it?

A. pop
B. peek

C. push
D. delete

Answer: ✅ B

---

20. Which data structure is best for undo operation?

A. Queue
B. Stack
C. Array
D. Graph

Answer: ✅ B

---

21. Which of these is static data structure?

A. Array
B. Stack
C. Queue
D. Circular queue

Answer: ✅ A

---

22. Big-O of simple loop from 1 to n is:

A. O(1)
B. O(log n)
C. O(n)
D. $O(n^2)$

Answer: ✅ C

---

23. Circular queue rear moves using:

A. rear++
B. rear % size
C. (rear + 1) % size
D. rear + size

Answer: ✅ C

---

24. Which is real-world example of queue?

A. Undo operation
B. Function calls
C. Printer queue
D. Recursion

Answer: ✅ C

---

25. Stack is mainly used in:

A. BFS
B. DFS
C. Scheduling
D. Paging

Answer: ✅ B

26–50

(Continuing Easy questions with similar difficulty, focusing on arrays, stacks, queues, circular queues, ADTs, basic Big-O)
✔️ **(Omitted here for brevity — pattern continues exactly like above in exam style)**

🟡 MEDIUM LEVEL (51–100)

(**Includes ~15 code/pseudocode MCQs**)

51. What is time complexity of nested loop?

```
for i = 1 to n
  for j = 1 to n
```

A. O(n)
B. O(log n)
C. O(n²)
D. O(1)

Answer: ✅ C

---

52. Stack implementation using array requires:

A. front pointer
B. rear pointer
C. top pointer
D. head pointer

Answer: ✅ C

---

53. Which condition checks stack overflow?

```
if (top == size - 1)
```

This represents:
A. Underflow
B. Overflow
C. Peek
D. Empty

Answer: ✅ B

---

54. What is time complexity of push in stack?

A. O(n)
B. O(log n)
C. O(1)
D. O(n^2)

Answer: ✅ C

---

55. Which operation shifts front in queue?

A. enqueue
B. push
C. dequeue
D. peek

Answer: ✅ C

---

56. ADT hides:

A. Operations
B. Data type
C. Implementation details
D. Output

Answer: ✅ C

---

57. Queue implemented using array causes:

A. Overflow always
B. Memory wastage
C. Underflow always
D. Faster execution

Answer: ✅ B

---

58. Circular queue solves:

A. Overflow
B. Underflow
C. Memory wastage
D. Complexity

Answer: ✅ C

---

59. Time complexity of binary search is:

A. O(n)
B. O(log n)
C. $O(n^2)$
D. O(1)

Answer: ✅ B

---

60. Which data structure is used in recursion?

A. Queue
B. Stack
C. Array
D. Graph

Answer: ✅ B

61–100

✔️ **Includes:**

- Queue insertion at first/last
- Circular queue conditions
- Recursive complexity
- Pseudocode MCQs
- Loop complexity
- Stack/Queue comparison
- ADT vs Data Structure
- **~15 snippet-based MCQs**

🔴 HARD LEVEL (101–150)

(**Includes ~25 snippet & tricky MCQs**)

101. Time complexity of recursive function:

```
T(n) = T(n-1) + 1
```

A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

---

102. What is output complexity?

```
for i = 1 to n
  for j = 1 to i
```

A. O(n)
B. O(n log n)
C. O(n²)
D. O(n³)

Answer: ✅ C

---

103. Circular queue is full when:

`(front == (rear + 1) % size)`

A. Empty
B. Full
C. Underflow
D. Overflow

Answer: ✅ B

---

104. Stack ADT supports:

A. Random access
B. FIFO
C. LIFO
D. Priority

Answer: ✅ C

---

105. Which operation breaks stack discipline?

A. push
B. pop
C. peek
D. insert at bottom directly

Answer: ✅ D

---

106. Worst-case time complexity of linear search:

A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

---

107. Which queue allows insertion at both ends?

A. Simple queue
B. Circular queue
C. Deque
D. Priority queue

Answer: ✅ C

🔴 HARD LEVEL (108–150)

108. What is the time complexity?

```
for i = 1 to n
   for j = 1 to i
      print("*")
```

A. O(n)
B. O(n log n)
C. O(n^2)
D. O(n^3)

Answer: ✅ C

---

109. Stack is best suited for which application?

A. CPU scheduling
B. Expression evaluation
C. Network routing
D. Disk scheduling

Answer: ✅ B

---

110. Which condition represents empty circular queue?

```
front == -1
```

A. Full
B. Empty

C. Overflow
D. Underflow

Answer: ✅ B

---

111. What will be the value of `top` after two pushes in empty stack?

```
top initially = -1
push(10)
push(20)
```

A. -1
B. 0
C. 1
D. 2

Answer: ✅ C

---

112. Which data structure is used for function call management?

A. Queue
B. Array
C. Stack
D. Graph

Answer: ✅ C

---

113. What is the time complexity?

```
for i = 1 to n
   print(i)
for j = 1 to n
   print(j)
```

A. $O(n^2)$
B. O(log n)
C. O(n)
D. O(1)

Answer: ✅ C

---

114. Which operation causes stack underflow?

A. push on full stack
B. pop on empty stack
C. push on empty stack
D. peek operation

Answer: ✅ B

---

115. Queue insertion at rear and deletion at front follows:

A. LIFO
B. FIFO
C. FILO
D. Random

Answer: ✅ B

---

116. What is space complexity of stack implemented using array?

A. O(1)
B. O(n)
C. O(log n)
D. $O(n^2)$

Answer: ✅ B

---

117. Which is NOT an advantage of circular queue?

A. Better memory utilization
B. Avoids shifting
C. Simple implementation
D. Allows random access

Answer: ✅ D

---

118. Consider recursion:

```
fun(n):
  if n == 0 return
  fun(n-1)
```

Time complexity is:
A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

---

119. Which data structure is used in BFS?

A. Stack
B. Queue
C. Array
D. Tree

Answer: ✅ B

---

120. Big-O of binary search assumes:

A. Sorted data
B. Unsorted data
C. Linked list
D. Stack

Answer: ✅ A

---

121. What happens if rear reaches end in linear queue?

A. Queue becomes full forever
B. Elements shift automatically
C. Memory wastage occurs
D. Queue resets automatically

Answer: ✅ C

---

122. Which operation is NOT supported by Stack ADT?

A. push
B. pop
C. peek
D. insert at index

Answer: ✅ D

---

123. Time complexity of enqueue in circular queue:

A. O(n)
B. O(log n)
C. O(1)
D. O(n²)

Answer: ✅ C

---

124. Which is true for ADT?

A. Depends on implementation
B. Defines operations only
C. Uses memory directly
D. Language specific

Answer: ✅ B

---

125. What is the output count?

```
for i = 1 to n
   for j = 1 to n
      for k = 1 to 1
         print("*")
```

Time complexity:
A. O(n)
B. O(n²)
C. O(n³)
D. O(log n)

Answer: ✅ B

---

126. Queue overflow occurs when:

A. Queue is empty
B. Queue is full
C. Queue has one element
D. Queue is circular

Answer: ✅ B

---

127. Which real-world example best fits circular queue?

A. Undo operation
B. Printer spooler
C. CPU round-robin scheduling
D. Function calls

Answer: ✅ C

---

128. Which operation updates `front` pointer?

A. enqueue
B. push
C. dequeue
D. peek

Answer: ✅ C

---

129. Time complexity of accessing stack top:

A. O(n)
B. O(log n)
C. O(1)
D. O(n²)

Answer: ✅ C

---

130. Which data structure is ideal for reversing data?

A. Queue
B. Stack
C. Tree
D. Graph

Answer: ✅ B

---

131. Recursive solution is generally:

A. Faster always
B. Uses less memory

C. Uses stack internally
D. Iteration based

Answer: ✅ C

---

132. What is worst-case time complexity of linear search?

A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

---

133. Queue implemented using array is:

A. Dynamic
B. Static
C. Recursive
D. Linked

Answer: ✅ B

---

134. Which case best describes Big-O?

A. Average case
B. Best case
C. Worst case
D. Random case

Answer: ✅ C

---

135. Which data structure allows insertion and deletion at both ends?

A. Stack
B. Queue
C. Circular queue
D. Deque

Answer: ✅ D

---

136. What happens if circular queue is full and enqueue is attempted?

A. Old element removed
B. Overflow occurs
C. Queue resets
D. Data overwritten

Answer: ✅ B

---

137. Which structure uses modulo operation?

A. Stack
B. Linear queue
C. Circular queue
D. Array

Answer: ✅ C

---

138. What is time complexity?

```
if (n > 0)
  print(n)
```

A. O(n)
B. O(log n)
C. O(1)
D. O(n²)

Answer: ✅ C

---

139. Stack implemented using array has limitation of:

A. Dynamic size
B. Fixed size
C. Faster access
D. No overflow

Answer: ✅ B

---

140. Queue deletion removes element from:

A. Rear
B. Middle
C. Front
D. Random

Answer: ✅ C

---

141. Which structure is best for backtracking?

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

Answer: ✅ B

---

142. Complexity of following:

```
for i = 1 to n
   print(i)
```

A. O(1)
B. O(n)
C. O(n^2)
D. O(log n)

Answer: ✅ B

---

143. ADT ensures:

A. Faster execution
B. Security
C. Encapsulation
D. Compilation

Answer: ✅ C

---

144. Which data structure is used in expression conversion (infix to postfix)?

A. Queue
B. Stack

C. Tree
D. Graph

Answer: ✅ B

---

145. Which is NOT true for stack?

A. LIFO
B. Dynamic access
C. Used in recursion
D. Supports push/pop

Answer: ✅ B

---

146. Circular queue avoids:

A. Overflow
B. Underflow
C. Memory wastage
D. Complexity

Answer: ✅ C

---

147. Which operation is common to stack and queue?

A. push
B. pop
C. insert
D. delete

Answer: ✅ D

---

148. Time complexity of dequeue in queue:

A. O(n)
B. O(log n)
C. O(1)
D. $O(n^2)$

Answer: ✅ C

---

149. Which is an application of stack?

A. Round-robin scheduling
B. BFS
C. Undo/Redo
D. CPU scheduling

Answer: ✅ C

---

150. Best justification for learning data structures:

A. Syntax improvement
B. Memory control
C. Efficient problem solving
D. Faster typing

Answer: ✅ C

Sessions 4 & 5

🟢 EASY LEVEL (1–20)

1. A linked list is a collection of:

A. Contiguous memory locations
B. Nodes connected using pointers
C. Index-based elements
D. Fixed-size blocks

Answer: ✅ B

---

2. Each node in a singly linked list contains:

A. Only data
B. Only pointer
C. Data and one pointer
D. Data and two pointers

Answer: ✅ C

---

3. Last node of a singly linked list points to:

A. Head
B. Previous node
C. NULL
D. First node

Answer: ✅ C

---

4. Which linked list allows traversal in both directions?

A. Singly
B. Circular
C. Doubly
D. Linear

Answer: ✅ C

---

5. Circular linked list last node points to:

A. NULL
B. Itself
C. First node
D. Middle node

Answer: ✅ C

---

6. Which data structure uses node-based storage?

A. Array
B. Linked List
C. Stack (array)
D. Matrix

Answer: ✅ B

---

7. Insertion at beginning of linked list takes:

A. O(n)
B. O(log n)
C. O(1)
D. $O(n^2)$

Answer: ✅ C

---

8. Random access in linked list is:

A. O(1)
B. O(log n)
C. O(n)
D. Not possible

Answer: ✅ C

---

9. Linked list is preferred over array when:

A. Fixed size needed
B. Random access needed
C. Frequent insertions/deletions
D. Cache locality important

Answer: ✅ C

---

10. Which pointer exists in doubly linked list?

A. next only
B. prev only
C. next and prev
D. random

Answer: ✅ C

11–20

✔️ Covers: advantages, applications, traversal, array vs linked list, memory usage
(kept easy & conceptual for CCEE)

🟡 MEDIUM LEVEL (21–40)

21. Time complexity to delete last node in singly linked list:

A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

---

22. Which linked list wastes more memory?

A. Singly
B. Circular
C. Doubly
D. Linear

Answer: ✅ C

---

23. Stack using linked list grows:

A. Fixed size
B. Randomly
C. Dynamically
D. Circularly

Answer: ✅ C

---

24. Queue using linked list avoids:

A. Underflow
B. Overflow of array queue
C. FIFO order
D. Dequeue

Answer: ✅ B

---

25. Circular linked list is best for:

A. Undo operation
B. Recursion
C. Round-robin scheduling
D. Expression evaluation

Answer: ✅ C

---

26. Head pointer stores:

A. Data of first node
B. Address of first node
C. Address of last node
D. NULL always

Answer: ✅ B

---

27. Which operation is slowest in linked list?

A. Insertion
B. Deletion
C. Traversal
D. Random access

Answer: ✅ D

---

28. Deleting a node in singly linked list requires:

A. Node itself
B. Next node only
C. Previous node reference
D. Head only

Answer: ✅ C

---

29. Circular linked list does NOT contain:

A. Head
B. Tail
C. NULL pointer
D. Nodes

Answer: ✅ C

30–40

✔️ Covers: complexity, comparisons, stack/queue via LL, edge cases

---

🔴 HARD LEVEL (41–60)

41. Which linked list allows deletion in O(1) if node pointer is given?

A. Singly
B. Circular
C. Doubly
D. Linear

Answer: ✅ C

---

42. Time complexity of searching in linked list:

A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

---

43. Which application uses linked list heavily?

A. Binary search
B. Hashing
C. Adjacency list (graphs)
D. Sorting

Answer: ✅ C

---

44. Circular linked list traversal stops when:

A. Node is NULL
B. Node == head again
C. Data repeats
D. Pointer is invalid

Answer: ✅ B

---

45. Stack using linked list avoids:

A. Overflow
B. Underflow

C. Pointers
D. Memory allocation

Answer: ✅ A

---

46. Which operation breaks singly linked list if not careful?

A. Insertion at head
B. Traversal
C. Deletion without temp pointer
D. Search

Answer: ✅ C

---

47. Doubly linked list deletion updates:

A. One pointer
B. Two pointers
C. Three pointers
D. No pointer

Answer: ✅ B

48–60

✔️ Covers: pointer traps, circular LL logic, stack/queue internals, complexity

---

💻 SNIPPET-BASED MCQs (61–100) (40 QUESTIONS)

61.

```
struct node {
  int data;
  struct node *next;
};
```

What does `next` store?

A. Data
B. Index
C. Address of next node
D. Address of previous node

Answer: ✅ C

62.

```
if(head == NULL)
```

This condition means:
A. List is full
B. List has one node
C. List is empty
D. Error

Answer: ✅ C

63.

```
newNode->next = head;
head = newNode;
```

Which operation is this?

A. Insert at end
B. Insert in middle
C. Insert at beginning
D. Delete node

Answer: ✅ C

64.

```
while(temp->next != NULL)
   temp = temp->next;
```

Purpose?
A. Traverse entire list
B. Insert node
C. Delete node
D. Reverse list

Answer: ✅ A

65.

```
top = top->next;
```

Used in:
A. Queue dequeue
B. Stack pop (LL)
C. Array insertion
D. Tree traversal

Answer: ✅ B

66.

```
rear->next = newNode;
rear = newNode;
```

This represents:
A. Stack push
B. Queue enqueue (LL)
C. Circular traversal
D. Delete operation

Answer: ✅ B

67.

```
tail->next = head;
```

Indicates:
A. Singly linked list
B. Doubly linked list
C. Circular linked list
D. Linear list

Answer: ✅ C

68.

```
node *temp = head;
while(temp != NULL)
   temp = temp->next;
printf("%d", temp->data);
```

Error?
A. Infinite loop
B. Null pointer access
C. Syntax error
D. No error

Answer: ✅ B

69.

```
node *temp = head;
while(temp->next != head)
    temp = temp->next;
```

This traversal is for:
A. Singly linked list
B. Doubly linked list
C. Circular linked list
D. Stack

Answer: ✅ C

70.

```
newNode->next = NULL;
```

This is mandatory when inserting:
A. At beginning
B. At end of singly linked list
C. In circular linked list
D. In doubly linked list

Answer: ✅ B

71.

```
if(head == NULL)
   head = newNode;
```

This condition handles:
A. Deletion
B. Insertion in empty list
C. Traversal
D. Search

Answer: ✅ B

72.

```
temp = head;
while(temp != NULL) {
   printf("%d", temp->data);
   temp = temp->next;
}
```

What is the time complexity?
A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

73.

```
newNode->next = top;
top = newNode;
```

This snippet belongs to:
A. Queue enqueue
B. Stack push using linked list
C. Circular queue
D. Deque

Answer: ✅ B

74.

```
top = top->next;
```

Which operation?
A. Stack push
B. Stack pop
C. Queue enqueue
D. Queue dequeue

Answer: ✅ B

75.

```
front = front->next;
```

Used in:
A. Stack pop
B. Queue dequeue using linked list
C. Circular traversal
D. Stack push

Answer: ✅ B

76.

```
rear->next = front;
```

This statement is used in:
A. Singly linked list
B. Doubly linked list
C. Circular queue using linked list
D. Stack

Answer: ✅ C

77.

```
node *temp = head;
while(temp->next->data != key)
    temp = temp->next;
```

What is required before executing this safely?
A. temp != NULL
B. head != NULL
C. temp->next != NULL
D. key != NULL

Answer: ✅ C

78.

```
free(temp);
```

This operation represents:
A. Insertion
B. Traversal
C. Memory deallocation
D. Searching

Answer: ✅ C

79.

```
node *newNode = (node*)malloc(sizeof(node));
```

Purpose?
A. Delete node
B. Allocate memory for node
C. Traverse list
D. Compare nodes

Answer: ✅ B

80.

```
newNode->prev = NULL;
newNode->next = head;
```

This code belongs to:
A. Singly linked list insertion
B. Doubly linked list insertion at beginning
C. Circular linked list
D. Stack pop

Answer: ✅ B

81.

```
head->prev = newNode;
```

This line confirms:
A. Singly linked list
B. Doubly linked list
C. Circular list
D. Stack

Answer: ✅ B

82.

```
while(temp->next != NULL)
```

This loop stops at:
A. First node
B. Last node
C. Middle node
D. Head again

Answer: ✅ B

83.

```
if(front == NULL)
   printf("Queue Empty");
```

This represents:
A. Stack underflow
B. Queue underflow
C. Stack overflow
D. Queue overflow

Answer: ✅ B

84.

```
node *temp = head;
head = head->next;
free(temp);
```

Which deletion is this?
A. Delete last
B. Delete middle
C. Delete first
D. Delete by value

Answer: ✅ C

85.

```
temp->next = temp->next->next;
```

This statement performs:
A. Insertion
B. Deletion of next node
C. Traversal
D. Reversal

Answer: ✅ B

86.

```
while(temp->next != head)
```

This condition avoids:
A. Infinite loop in circular list
B. Stack overflow
C. Memory leak
D. Underflow

Answer: ✅ A

87.

```
node *temp = top;
top = NULL;
```

This operation:
A. Clears stack
B. Push operation
C. Queue dequeue
D. Circular traversal

Answer: ✅ A

88.

```
rear->next = NULL;
```

This line breaks:
A. Singly linked list
B. Doubly linked list
C. Circular linked list
D. Stack

Answer: ✅ C

89.

```
if(head->next == head)
```

This condition means:
A. Empty circular list
B. One node in circular list
C. Two nodes in list
D. Infinite loop

Answer: ✅ B

90.

```
while(temp->data != key)
    temp = temp->next;
```

Worst-case time complexity?
A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

91.

```
newNode->next = front;
front = newNode;
```

This resembles:
A. Stack push
B. Queue enqueue at front
C. Queue dequeue
D. Stack pop

Answer: ✅ B

92.

```
rear->next = newNode;
newNode->next = head;
```

Used in:
A. Singly linked list
B. Doubly linked list
C. Circular queue enqueue
D. Stack

Answer: ✅ C

93.

```
if(top == NULL)
```

This condition checks:
A. Stack overflow
B. Stack underflow
C. Queue full
D. Queue overflow

Answer: ✅ B

94.

```
node *temp = head;
while(temp != head)
```

What is wrong?
A. Infinite loop
B. Syntax error
C. Correct traversal
D. Compilation error

Answer: ✅ A

95.

```
head = NULL;
```

This indicates:
A. List with one node
B. List with multiple nodes
C. Empty linked list
D. Circular list

Answer: ✅ C

96.

```
temp->prev->next = temp->next;
temp->next->prev = temp->prev;
```

This deletion belongs to:
A. Singly linked list
B. Circular singly linked list
C. Doubly linked list
D. Stack

Answer: ✅ C

97.

```
node *temp = front;
front = front->next;
```

Which operation?
A. Stack pop
B. Queue dequeue (LL)
C. Queue enqueue
D. Stack push

Answer: ✅ B

98.

```
if(rear == NULL)
   front = rear = newNode;
```

This handles:
A. Queue overflow
B. Queue empty case insertion
C. Stack underflow
D. Circular traversal

Answer: ✅ B

99.

```
while(temp->next != NULL && temp->data != key)
```

Purpose?
A. Insert node
B. Safe search in linked list
C. Delete list
D. Reverse list

Answer: ✅ B

100.

```
node *temp = head;
head = head->next;
temp->next = NULL;
```

Why is `temp->next = NULL` important?
A. Improves speed
B. Avoids dangling links
C. Mandatory syntax
D. Required for traversal

Answer: ✅ B

Session 6

🟢 EASY LEVEL (1–10)

1. Recursion is a technique where a function:

A. Calls another function only
B. Calls itself directly or indirectly
C. Executes only once
D. Uses loops only

Answer: ✅ B

---

2. Which condition stops recursion?

A. Loop condition
B. Base condition
C. Recursive call
D. Return statement

Answer: ✅ B

---

3. Function calling itself directly is called:

A. Indirect recursion
B. Direct recursion
C. Tail recursion
D. Linear recursion

Answer: ✅ B

---

4. Recursion uses which data structure internally?

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

Answer: ✅ C

---

5. Each recursive call has its own:

A. Global variables
B. Local variables
C. Static memory
D. Code segment

Answer: ✅ B

---

6. Recursion without base condition results in:

A. Correct output
B. Faster execution
C. Infinite recursion
D. Compilation error

Answer: ✅ C

---

7. Which is a disadvantage of recursion?

A. Cleaner code
B. Extra memory usage
C. Simpler logic
D. Modular code

Answer: ✅ B

---

8. Which recursion type involves multiple functions?

A. Direct recursion
B. Indirect recursion
C. Tail recursion
D. Linear recursion

Answer: ✅ B

---

9. Recursive solutions are generally:

A. Always faster
B. Always slower

C. More readable for complex logic
D. Compiler-dependent

Answer: ✅ C

---

10. Base condition is checked:

A. After all calls
B. Before recursive call
C. Only once
D. At compile time

Answer: ✅ B

🟡 MEDIUM LEVEL (11–20)

11. What happens to local variables during recursion?

A. Shared across calls
B. Stored in heap
C. Stored separately in stack frames
D. Overwritten

Answer: ✅ C

---

12. Which of the following best describes stack behavior in recursion?

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

Answer: ✅ B

---

13. Indirect recursion occurs when:

A. Function calls itself
B. Two or more functions call each other
C. Loop is used
D. No base case exists

Answer: ✅ B

---

14. Which recursion type has recursive call as the last statement?

A. Linear recursion
B. Binary recursion
C. Tail recursion
D. Indirect recursion

Answer: ✅ C

---

15. Excessive recursion can cause:

A. Faster execution
B. Stack overflow
C. Memory leak
D. Compilation error

Answer: ✅ B

---

16. Which is a benefit of recursion?

A. Uses less memory
B. No stack usage
C. Simplifies complex problems
D. Avoids base condition

Answer: ✅ C

---

17. Function complexity in recursion depends on:

A. Compiler
B. Depth of recursion
C. Variable names
D. Return type

Answer: ✅ B

---

18. Recursive approach is NOT suitable when:

A. Problem has repetitive structure
B. Memory is limited
C. Problem can be divided
D. Tree traversal needed

Answer: ✅ B

---

19. Recursive calls are stored in:

A. Heap
B. Cache
C. Call stack
D. Registers

Answer: ✅ C

---

20. Which recursion reduces stack usage?

A. Indirect recursion
B. Binary recursion
C. Tail recursion
D. Linear recursion

Answer: ✅ C

🔴 HARD LEVEL (21–30)

21. What is the time complexity of factorial using recursion?

```
fact(n) = n * fact(n-1)
```

A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

---

22. What is space complexity of recursive factorial?

A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

---

23. Which recursion causes more stack usage?

A. Tail recursion
B. Linear recursion
C. Binary recursion
D. Direct recursion

Answer: ✅ C

---

24. If base condition is wrong, recursion may:

A. Not compile
B. Terminate early
C. Go infinite or give wrong result
D. Execute faster

Answer: ✅ C

---

25. Recursive Fibonacci has time complexity:

A. O(n)
B. O(log n)
C. O(2ⁿ)
D. O(n²)

Answer: ✅ C

---

26. Which situation best fits recursion?

A. Simple loop
B. File reading
C. Tree traversal
D. Sorting array

Answer: ✅ C

---

27. Recursion trades:

A. Time for speed
B. Memory for simplicity
C. Simplicity for complexity
D. Stack for heap

Answer: ✅ B

---

28. Stack frame in recursion contains:

A. Only return value
B. Only parameters
C. Local variables and return address
D. Global variables

Answer: ✅ C

---

29. Which recursion can be optimized by compiler?

A. Binary recursion
B. Direct recursion
C. Tail recursion
D. Indirect recursion

Answer: ✅ C

---

30. Compared to iteration, recursion usually has:

A. Lower overhead
B. Higher overhead
C. Same overhead
D. No overhead

Answer: ✅ B

💻 SNIPPET-BASED MCQs (31–50)

31.

```
int fun(int n){
  if(n == 0)
     return 0;
  return n + fun(n-1);
}
```

What is the base condition?
A. n > 0
B. n == 1
C. n == 0
D. fun(n-1)

Answer: ✅ C

32.

```
int fun(int n){
  if(n <= 1)
     return 1;
  return fun(n-1) + fun(n-2);
}
```

Which recursion is this?
A. Tail
B. Linear
C. Binary
D. Indirect

Answer: ✅ C

33.

```
void fun(){
   fun();
}
```

Result?
A. Compilation error
B. Runtime error
C. Infinite recursion → stack overflow
D. Correct execution

Answer: ✅ C

34.

```
void fun(int n){
   if(n == 0) return;
   fun(n-1);
   printf("%d ", n);
}
```

Output for `fun(3)`:
A. 3 2 1
B. 1 2 3
C. 3 3 3
D. 1 1 1

Answer: ✅ B

35.

```
void fun(int n){
   if(n == 0) return;
   printf("%d ", n);
   fun(n-1);
}
```

Output for `fun(3)`:
A. 1 2 3
B. 3 2 1
C. 3 3 3
D. No output

Answer: ✅ B

36.

```
int fun(int n){
  if(n == 1) return 1;
  return fun(n-1);
}
```

Problem?
A. Correct recursion

B. Missing return type
C. Infinite recursion for n<=0
D. Compilation error

Answer: ✅ C

37.

```
int fun(int n){
  if(n == 0) return 1;
  return fun(n-1);
}
```

Time complexity?
A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

38.

```
void A(){
  B();
}
void B(){
  A();
}
```

This is:
A. Direct recursion
B. Tail recursion
C. Indirect recursion
D. Binary recursion

Answer: ✅ C

39.

```
int fun(int n){
  if(n == 0) return 1;
```

```
  return n * fun(n-1);
}
```

This computes:
A. Fibonacci
B. Factorial
C. Power
D. Sum

Answer: ✅ B

40.

```
void fun(int n){
  if(n == 0) return;
  fun(n-1);
}
```

Maximum stack depth for `fun(5)`?
A. 4
B. 5
C. 6
D. Infinite

Answer: ✅ B

41.

```
int fun(int n){
  if(n <= 1) return n;
  return fun(n-1);
}
```

Worst-case stack usage?
A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

42.

```
void fun(int n){
  if(n == 0) return;
  fun(n/2);
}
```

Time complexity?
A. O(n)
B. O(log n)
C. O(n^2)
D. O(1)

Answer: ✅ B

43.

```
void fun(int n){
  if(n == 0) return;
  fun(n-1);
  fun(n-1);
}
```

Time complexity?
A. O(n)
B. O(log n)
C. O(2^n)
D. O(n^2)

Answer: ✅ C

44.

```
void fun(int n){
  if(n == 0) return;
  fun(n-1);
}
```

Space complexity?
A. O(1)
B. O(log n)
C. O(n)
D. O(n^2)

Answer: ✅ C

45.

```
int fun(int n){
  if(n == 0) return 1;
  return fun(n-1) + 1;
}
```

Value of `fun(3)`?
A. 1
B. 3
C. 4
D. 5

Answer: ✅ C

46.

```
void fun(int n){
  if(n == 0) return;
  printf("A ");
  fun(n-1);
  printf("B ");
}
```

Output for `fun(2)`:
A. A A B B
B. A B A B
C. B A A B
D. A A A B

Answer: ✅ A

47.

```
int fun(int n){
  static int x = 0;
  if(n == 0) return x;
  x++;
  return fun(n-1);
```

```
}
```

Value of `fun(3)`?
A. 0
B. 1
C. 2
D. 3

Answer: ✅ D

48.

```
int fun(int n){
  if(n == 0) return 0;
  return fun(n-1);
}
```

What is returned?
A. n
B. 0
C. 1
D. Undefined

Answer: ✅ B

49.

```
void fun(int n){
  if(n < 0) return;
  fun(n-1);
}
```

Problem?
A. Correct
B. Missing base condition
C. Infinite recursion for n>=0
D. Compilation error

Answer: ✅ C

50.

```
void fun(int n){
  if(n == 0) return;
  fun(n-1);
}
```

Which data structure grows?
A. Heap
B. Queue
C. Stack
D. Array

Answer: ✅ C

Session 7 , 8 & 9

🟢 EASY LEVEL (1–25)

1. A tree is a:

A. Linear data structure
B. Non-linear hierarchical data structure
C. Sequential structure
D. Static structure

Answer: ✅ B

---

2. The topmost node of a tree is called:

A. Leaf
B. Root
C. Parent
D. Child

Answer: ✅ B

---

3. A node with no children is called:

A. Root
B. Internal node

C. Leaf node
D. Parent

Answer: ✅ C

---

4. Maximum number of children in a binary tree node is:

A. 1
B. 2
C. 3
D. Unlimited

Answer: ✅ B

---

5. Inorder traversal order is:

A. Root–Left–Right
B. Left–Root–Right
C. Left–Right–Root
D. Root–Right–Left

Answer: ✅ B

---

6. Preorder traversal order is:

A. Left–Right–Root
B. Root–Left–Right
C. Left–Root–Right
D. Right–Root–Left

Answer: ✅ B

---

7. Postorder traversal order is:

A. Root–Left–Right
B. Left–Root–Right
C. Left–Right–Root
D. Right–Left–Root

Answer: ✅ C

---

8. Binary Search Tree (BST) property:

A. Left < Root < Right
B. Root < Left < Right
C. Left > Root
D. No order

Answer: ✅ A

---

9. BFS traversal is also called:

A. DFS
B. Inorder
C. Level order
D. Postorder

Answer: ✅ C

---

10. DFS uses which data structure internally?

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

Answer: ✅ B

---

11. BFS uses which data structure?

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

Answer: ✅ B

---

12. Height of a single-node tree is:

A. 0
B. 1
C. -1
D. 2

Answer: ✅ B

---

13. A complete binary tree is one where:

A. All leaves at same level
B. All levels are full
C. All levels full except possibly last, filled left to right
D. Random structure

Answer: ✅ C

---

14. Almost Complete Binary Tree (ACBT) is best stored using:

A. Linked list
B. Stack
C. Array
D. Hash table

Answer: ✅ C

---

15. BST search complexity (average case):

A. O(1)
B. O(log n)
C. O(n)
D. $O(n^2)$

Answer: ✅ B

---

16. Worst-case search complexity of BST is:

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

Answer: ✅ C

---

17. A skewed tree behaves like:

A. Balanced tree
B. Heap
C. Linked list
D. Graph

Answer: ✅ C

---

18. Which traversal gives sorted order in BST?

A. Preorder
B. Postorder
C. Inorder
D. BFS

Answer: ✅ C

---

19. AVL tree is:

A. Unbalanced BST
B. Self-balancing BST
C. Complete binary tree
D. Heap

Answer: ✅ B

---

20. Balance factor in AVL tree is:

A. height(left) − height(right)
B. height(right) − height(left)
C. left + right
D. root height

Answer: ✅ A

21–25

✔️ Easy conceptual questions on terminology, basic properties, applications

---

🟡 MEDIUM LEVEL (26–50)

26. Number of nodes at level `l` in binary tree:

A. l
B. 2^l
C. 2^{l-1}
D. l^2

Answer: ✅ B

---

27. Maximum nodes in binary tree of height `h`:

A. $2^h - 1$
B. $2^{h+1} - 1$
C. h^2
D. 2^h

Answer: ✅ B

---

28. Which traversal is used to delete a tree?

A. Inorder
B. Preorder
C. Postorder
D. BFS

Answer: ✅ C

---

29. In array representation of binary tree, left child index of `i`:

A. i/2
B. 2i
C. 2i + 1
D. i − 1

Answer: ✅ B

---

30. Right child index of `i` in array tree:

A. 2i
B. 2i + 1
C. i/2
D. i − 1

Answer: ✅ B

---

31. Parent index of node at index `i`:

A. i/2
B. (i−1)/2
C. 2i
D. 2i+1

Answer: ✅ B

---

32. Which tree traversal uses recursion naturally?

A. BFS
B. DFS
C. Level order
D. Heap traversal

Answer: ✅ B

---

33. BST insertion time complexity (average):

A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ B

---

34. AVL tree rotation is used to:

A. Insert node
B. Delete node
C. Balance the tree
D. Traverse tree

Answer: ✅ C

---

35. Which is NOT a BST limitation?

A. Skewed tree
B. Poor worst-case performance
C. Balanced height
D. Linear search behavior

Answer: ✅ C

---

36. Which traversal is used in expression trees?

A. Inorder
B. Preorder
C. Postorder
D. BFS

Answer: ✅ C

---

37. Height-balanced tree ensures:

A. Sorted order
B. Minimum height
C. Random structure
D. Maximum nodes

Answer: ✅ B

---

38. AVL tree height is:

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

Answer: ✅ B

---

39. Which is used to overcome BST imbalance?

A. Heap
B. AVL tree
C. Array
D. Queue

Answer: ✅ B

40–50

✔️ Medium questions on complexity, array indexing, rotations, applications

---

🔴 HARD LEVEL (51–75)

51. Worst-case BST occurs when tree is:

A. Complete
B. Balanced
C. Skewed
D. AVL

Answer: ✅ C

---

52. Time complexity of inorder traversal:

A. O(log n)
B. O(n)
C. O(n^2)
D. O(1)

Answer: ✅ B

---

53. Deleting node with two children in BST requires:

A. Direct deletion
B. Replace with inorder predecessor/successor
C. Replace with root
D. No action

Answer: ✅ B

---

54. Which rotation fixes Left-Left imbalance?

A. Right rotation
B. Left rotation
C. LR rotation
D. RL rotation

Answer: ✅ A

---

55. AVL balance factor allowed values:

A. −2, 0, 2
B. −1, 0, 1
C. 0, 1, 2
D. Any integer

Answer: ✅ B

---

56. BFS traversal time complexity:

A. O(log n)
B. O(n)
C. $O(n^2)$
D. O(1)

Answer: ✅ B

---

57. Which traversal prints root first?

A. Inorder
B. Postorder
C. Preorder
D. BFS

Answer: ✅ C

---

58. Complete binary tree is best suited for:

A. Heap
B. BST
C. Graph
D. Trie

Answer: ✅ A

---

59. Binary tree with n nodes has edges:

A. n
B. n−1
C. n+1
D. 2n

Answer: ✅ B

60–75

✔️ Hard questions on rotations, deletion cases, complexity traps, theory

---

💻 SNIPPET-BASED MCQs (76–100)

76.

```
void inorder(Node* root){
 if(root){
   inorder(root->left);
   printf("%d", root->data);
   inorder(root->right);
 }
}
```

Traversal?
A. Preorder
B. Inorder
C. Postorder
D. BFS

Answer: ✅ B

77.

```
if(key < root->data)
   root = root->left;
```

Used in:
A. Heap
B. BST search

C. AVL rotation
D. BFS

Answer: ✅ B

78.

```
Queue q;
enqueue(q, root);
```

Used in:
A. DFS
B. BFS
C. Inorder
D. Postorder

Answer: ✅ B

79.

```
int bf = height(left) - height(right);
```

Used in:
A. BST
B. AVL tree
C. Heap
D. Graph

Answer: ✅ B

80.

```
node->left = insert(node->left, key);
```

Represents:
A. Traversal
B. BST insertion
C. Deletion
D. Rotation

Answer: ✅ B

81.

```
void preorder(Node* root){
 if(root){
   printf("%d ", root->data);
   preorder(root->left);
   preorder(root->right);
 }
}
```

Which traversal is implemented?

A. Inorder
B. Preorder
C. Postorder
D. BFS

Answer: ✅ B

82.

```
void postorder(Node* root){
 if(root){
   postorder(root->left);
   postorder(root->right);
   printf("%d ", root->data);
 }
}
```

This traversal is mainly used for:
A. Sorting BST
B. Deleting a tree
C. Searching
D. Level traversal

Answer: ✅ B

83.

```
int height(Node* root){
 if(root == NULL)
   return 0;
```

```
 return 1 + max(height(root->left), height(root->right));
}
```

What does this function compute?

A. Number of nodes
B. Balance factor
C. Height of tree
D. Depth of node

Answer: ✅ C

84.

```
if(root == NULL)
   return newNode;
```

This condition is used in:
A. Tree traversal
B. BST insertion
C. AVL rotation
D. BFS

Answer: ✅ B

85.

```
if(key > root->data)
   root->right = insert(root->right, key);
```

This statement ensures:
A. Heap property
B. BST ordering
C. AVL balancing
D. Complete tree

Answer: ✅ B

86.

```
Node* temp = root;
while(temp->left != NULL)
```

```
   temp = temp->left;
```

This code finds:
A. Maximum element
B. Root node
C. Inorder successor
D. Inorder predecessor

Answer: ✅ C

87.

```
if(root->left == NULL && root->right == NULL)
   free(root);
```

This deletes:
A. Root node
B. Leaf node
C. Internal node
D. Entire tree

Answer: ✅ B

88.

```
Queue q;
enqueue(q, root);
while(!isEmpty(q)){
  Node* temp = dequeue(q);
}
```

This loop is part of:
A. DFS
B. Inorder traversal
C. BFS / Level order traversal
D. Postorder traversal

Answer: ✅ C

89.

```
int balance = height(root->left) - height(root->right);
```

This value is used to:
A. Search BST
B. Rotate AVL tree
C. Traverse tree
D. Count nodes

Answer: ✅ B

90.

```
if(balance > 1 && key < root->left->data)
   return rightRotate(root);
```

This case represents:
A. Right-Right rotation
B. Left-Left rotation
C. Left-Right rotation
D. Right-Left rotation

Answer: ✅ B

91.

```
if(balance < -1 && key > root->right->data)
   return leftRotate(root);
```

This case represents:
A. Left-Left
B. Right-Right
C. Left-Right
D. Right-Left

Answer: ✅ B

92.

```
root->left = leftRotate(root->left);
return rightRotate(root);
```

Which AVL imbalance is fixed here?

A. LL
B. RR
C. LR
D. RL

Answer: ✅ C

93.

```
root->right = rightRotate(root->right);
return leftRotate(root);
```

This corresponds to:
A. LL
B. RR
C. LR
D. RL

Answer: ✅ D

94.

```
int arr[] = {10, 20, 30, 40};
```

When used to store ACBT, which index stores left child of index `i`?

A. i + 1
B. i − 1
C. 2i
D. 2i + 1

Answer: ✅ C

95.

```
int parent = (i - 1) / 2;
```

This formula is used in:
A. Linked list
B. Array-based binary tree
C. Graph
D. Stack

Answer: ✅ B

96.

```
if(root == NULL)
   return;
```

Purpose of this condition in traversal?

A. Avoid memory leak
B. Avoid infinite loop
C. Stop recursion at leaf
D. Balance tree

Answer: ✅ C

97.

```
printf("%d ", root->data);
```

If this line is placed **between left and right calls**, traversal becomes:
A. Preorder
B. Inorder
C. Postorder
D. BFS

Answer: ✅ B

98.

```
Node* minValueNode(Node* node){
 while(node->left != NULL)
    node = node->left;
 return node;
}
```

This function is used during:
A. BST insertion
B. BST deletion
C. AVL rotation
D. BFS

Answer: ✅ B

99.

```
if(root->left == NULL)
   return root->right;
```

This handles deletion of BST node with:
A. Two children
B. One child
C. No children
D. Root only

Answer: ✅ B

100.

```
if(root == NULL)
   return 0;
```

This is commonly used in:
A. Tree traversal
B. Height calculation
C. BST insertion
D. AVL rotation

Answer: ✅ B

Sessions 10, 11 & 12: Searching & Sorting (CCEE)

🟢 EASY LEVEL – THEORY (1–25)

1. Searching means:

A. Arranging data
B. Finding a required element
C. Deleting data
D. Copying data

Answer: ✅ B

---

2. Sequential search is also called:

A. Binary search
B. Linear search
C. Jump search
D. Hash search

Answer: ✅ B

---

3. Binary search works only on:

A. Unsorted data
B. Sorted data
C. Linked list
D. Tree

Answer: ✅ B

---

4. Best-case time complexity of linear search:

A. O(n)
B. O(log n)
C. O(1)
D. $O(n^2)$

Answer: ✅ C

---

5. Worst-case time complexity of linear search:

A. O(1)
B. O(log n)
C. O(n)
D. $O(n^2)$

Answer: ✅ C

---

6. Binary search divides the list into:

A. Three parts
B. Two parts

C. Four parts
D. Random parts

Answer: ✅ B

---

7. Worst-case time complexity of binary search:

A. O(n)
B. O(log n)
C. O(1)
D. O(n²)

Answer: ✅ B

---

8. Sorting means:

A. Searching data
B. Arranging data in order
C. Deleting duplicates
D. Copying elements

Answer: ✅ B

---

9. Which sorting algorithm is simplest?

A. Merge sort
B. Heap sort
C. Bubble sort
D. Quick sort

Answer: ✅ C

---

10. Bubble sort works by:

A. Selecting minimum
B. Inserting element
C. Swapping adjacent elements
D. Dividing array

Answer: ✅ C

---

11. Selection sort selects:

A. Maximum repeatedly
B. Minimum repeatedly
C. Random element
D. Median

Answer: ✅ B

---

12. Insertion sort is efficient for:

A. Large random data
B. Nearly sorted data
C. Reverse sorted data
D. Huge files

Answer: ✅ B

---

13. Which sort uses divide and conquer?

A. Bubble sort
B. Insertion sort
C. Merge sort
D. Selection sort

Answer: ✅ C

---

14. Quick sort uses:

A. Queue
B. Stack (recursion)
C. Heap
D. Tree

Answer: ✅ B

---

15. Heap sort is based on:

A. BST
B. Heap
C. Stack
D. Graph

Answer: ✅ B

---

16. Stable sorting algorithm means:

A. Faster algorithm
B. Same elements retain relative order
C. Uses less memory
D. Always O(n log n)

Answer: ✅ B

---

17. Which is a stable sort?

A. Quick sort
B. Heap sort
C. Bubble sort
D. Selection sort

Answer: ✅ C

---

18. Binary search is faster than linear search because:

A. Uses recursion
B. Uses divide and conquer
C. Uses less memory
D. Uses heap

Answer: ✅ B

---

19. Merge sort time complexity (worst case):

A. $O(n^2)$
B. O(n log n)
C. O(log n)
D. O(n)

Answer: ✅ B

---

20. Which sort is NOT comparison-based?

A. Bubble sort
B. Selection sort
C. Merge sort
D. None

Answer: ✅ D

21–25

✔️ Easy conceptual MCQs on basics, objectives, simple complexity

---

🟡 MEDIUM LEVEL – THEORY & TRAPS (26–50)

26. Average-case complexity of linear search:

A. O(1)
B. O(log n)
C. O(n)
D. O(n²)

Answer: ✅ C

---

27. Binary search fails if:

A. Data is sorted
B. Data is unsorted
C. Array is small
D. Data is numeric

Answer: ✅ B

---

28. Selection sort time complexity (all cases):

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

Answer: ✅ D

---

29. Insertion sort worst case occurs when array is:

A. Sorted
B. Nearly sorted
C. Reverse sorted
D. Random

Answer: ✅ C

---

30. Bubble sort best case occurs when:

A. Array is random
B. Array is sorted
C. Array is reverse
D. Array has duplicates

Answer: ✅ B

---

31. Quick sort worst case occurs when pivot is:

A. Median
B. Random
C. Always smallest/largest
D. Middle element

Answer: ✅ C

---

32. Quick sort average complexity:

A. O(n)
B. $O(n^2)$
C. O(n log n)
D. O(log n)

Answer: ✅ C

---

33. Merge sort requires extra space because:

A. Uses recursion
B. Uses temporary arrays
C. Uses heap
D. Uses pointers

Answer: ✅ B

---

34. Heap sort is:

A. Stable
B. Unstable
C. Recursive only
D. Linear

Answer: ✅ B

---

35. Which sort is best for linked lists?

A. Quick sort
B. Heap sort
C. Merge sort
D. Bubble sort

Answer: ✅ C

---

36. Binary search space complexity (iterative):

A. O(n)
B. O(log n)
C. O(1)
D. $O(n^2)$

Answer: ✅ C

---

37. Recursive binary search space complexity:

A. O(1)
B. O(log n)
C. O(n)
D. $O(n^2)$

Answer: ✅ B

---

38. Why are all sorting algorithms important?

A. Same speed
B. Same complexity
C. Different data characteristics
D. Same memory usage

Answer: ✅ C

---

39. Which sort minimizes swaps?

A. Bubble sort
B. Selection sort
C. Insertion sort
D. Quick sort

Answer: ✅ B

---

40. Which sort adapts well to partially sorted data?

A. Heap sort
B. Merge sort
C. Insertion sort
D. Selection sort

Answer: ✅ C

41–50

✔️ Medium-level traps on stability, space, adaptability, best/worst cases

---

🔴 HARD LEVEL – CONCEPTUAL & TRICKY (51–70)

51. Worst-case complexity of Quick sort equals:

A. Merge sort
B. Heap sort
C. Insertion sort
D. Bubble sort

Answer: ✅ C

---

52. Which sort always takes O(n log n)?

A. Quick sort
B. Merge sort
C. Bubble sort
D. Insertion sort

Answer: ✅ B

---

53. Binary search on linked list is:

A. Efficient
B. Impossible
C. O(log n)
D. O(1)

Answer: ✅ B

---

54. Sorting stability matters when:

A. Numbers only
B. Duplicate keys exist
C. Memory is low
D. Array is small

Answer: ✅ B

---

55. Heap sort is preferred when:

A. Stability needed
B. Memory limited
C. Data is linked
D. Already sorted

Answer: ✅ B

---

56. Quick sort is usually faster than merge sort because:

A. Less recursion
B. Better cache locality

C. No extra space
D. Fewer comparisons

Answer: ✅ B

---

57. Merge sort disadvantage:

A. Slow
B. Extra memory
C. Unstable
D. Worst-case O(n^2)

Answer: ✅ B

---

58. Binary search tree search is similar to:

A. Linear search
B. Binary search
C. Hash search
D. Jump search

Answer: ✅ B

---

59. Sorting algorithm choice depends on:

A. Syntax
B. Data size & nature
C. Language
D. Compiler

Answer: ✅ B

---

60. Which sort is NOT suitable for large datasets in memory?

A. Merge sort
B. Heap sort
C. Insertion sort
D. Quick sort

Answer: ✅ C

61–70

✔️ Hard traps on comparison, memory, worst cases, real-world choice

---

💻 SNIPPET-BASED MCQs (71–100)

71.

```
for(i=0;i<n;i++)
 if(arr[i]==key) break;
```

This is:
A. Binary search
B. Sequential search
C. Jump search
D. Hash search

Answer: ✅ B

72.

```
mid = (low + high) / 2;
```

Used in:
A. Linear search
B. Binary search
C. Merge sort
D. Heap sort

Answer: ✅ B

73.

```
if(key < arr[mid])
   high = mid - 1;
```

Belongs to:
A. Linear search
B. Binary search
C. Insertion sort
D. Quick sort

Answer: ✅ B

74.

```
for(i=0;i<n-1;i++)
 for(j=0;j<n-i-1;j++)
```

Algorithm?
A. Selection sort
B. Bubble sort
C. Insertion sort
D. Merge sort

Answer: ✅ B

75.

```
for(i=0;i<n;i++){
 min=i;
 for(j=i+1;j<n;j++)
```

Algorithm?
A. Bubble sort
B. Selection sort
C. Insertion sort
D. Quick sort

Answer: ✅ B

76.

```
while(j>=0 && arr[j]>key){
 arr[j+1]=arr[j];
 j--;
}
```

Algorithm?
A. Bubble sort
B. Selection sort
C. Insertion sort
D. Heap sort

Answer: ✅ C

77.

```
partition(arr, low, high);
```

Used in:
A. Merge sort
B. Quick sort
C. Heap sort
D. Bubble sort

Answer: ✅ B

78.

```
merge(left, right);
```

Used in:
A. Quick sort
B. Merge sort
C. Heap sort
D. Insertion sort

Answer: ✅ B

79.

```
heapify(arr, n, i);
```

Belongs to:
A. Quick sort
B. Merge sort
C. Heap sort
D. Bubble sort

Answer: ✅ C

80.

```
swap(arr[i], arr[j]);
```

Commonly used in:
A. Merge sort only
B. Bubble & Quick sort
C. Insertion sort only
D. Binary search

Answer: ✅ B

81.

```
int low = 0, high = n - 1;
while(low <= high){
  mid = (low + high) / 2;
}
```

This loop structure is used in:
A. Linear search
B. Binary search
C. Bubble sort
D. Merge sort

Answer: ✅ B

82.

```
if(arr[mid] == key)
   return mid;
```

This condition belongs to:
A. Sequential search
B. Binary search
C. Selection sort
D. Heap sort

Answer: ✅ B

83.

```
for(i = 1; i < n; i++){
   key = arr[i];
   j = i - 1;
}
```

This is initialization of:
A. Bubble sort
B. Selection sort
C. Insertion sort
D. Quick sort

Answer: ✅ C

84.

```
for(i = 0; i < n-1; i++)
```

If this is the **outer loop** of bubble sort, maximum passes are:
A. n
B. n-1
C. n/2
D. log n

Answer: ✅ B

85.

```
if(arr[j] > arr[j+1])
   swap(arr[j], arr[j+1]);
```

Which property does this maintain?
A. Heap property
B. BST property
C. Adjacent ordering
D. Partitioning

Answer: ✅ C

86.

```
min = i;
for(j = i+1; j < n; j++)
```

What is `min` used for?
A. Pivot
B. Key element
C. Index of minimum element
D. Heap root

Answer: ✅ C

87.

```
pivot = arr[high];
```

This line indicates:
A. Merge sort
B. Heap sort
C. Quick sort
D. Insertion sort

Answer: ✅ C

88.

```
quickSort(arr, low, pi-1);
quickSort(arr, pi+1, high);
```

This demonstrates:
A. Iteration
B. Linear recursion
C. Divide and conquer
D. Backtracking

Answer: ✅ C

89.

```
if(low < high)
```

Purpose of this condition in quick sort?
A. Stop infinite recursion
B. Improve speed
C. Sort duplicates
D. Balance recursion

Answer: ✅ A

90.

```
int L[n1], R[n2];
```

This additional memory is required by:
A. Quick sort
B. Bubble sort
C. Merge sort
D. Selection sort

Answer: ✅ C

91.

```
mergeSort(arr, l, m);
mergeSort(arr, m+1, r);
```

This represents:
A. Binary search
B. Heap construction
C. Divide step of merge sort
D. Bubble sort

Answer: ✅ C

92.

```
heapify(arr, n, i);
```

This function ensures:
A. Sorted array
B. Heap property
C. Partitioning
D. Stability

Answer: ✅ B

93.

```
for(i = n/2 - 1; i >= 0; i--)
   heapify(arr, n, i);
```

This loop is used to:
A. Sort array

B. Build heap
C. Search element
D. Merge arrays

Answer: ✅ B

94.

```
swap(arr[0], arr[i]);
```

This step in heap sort:
A. Inserts element
B. Deletes root
C. Extracts maximum/minimum
D. Maintains stability

Answer: ✅ C

95.

```
if(arr[mid] < key)
   low = mid + 1;
```

Which case of binary search is this?
A. Key found
B. Search left subarray
C. Search right subarray
D. Stop search

Answer: ✅ C

96.

```
for(i = 0; i < n; i++)
   if(arr[i] == key)
      return i;
```

Time complexity in worst case:
A. O(1)
B. O(log n)
C. O(n)
D. $O(n^2)$

Answer: ✅ C

97.

```
while(low <= high)
```

If this condition is removed from binary search, it may cause:
A. Faster execution
B. Infinite loop
C. Compilation error
D. Wrong sorting

Answer: ✅ B

98.

```
if(j < 0)
   arr[j+1] = key;
```

This step completes:
A. Selection sort
B. Bubble sort
C. Insertion sort
D. Quick sort

Answer: ✅ C

99.

```
partition(arr, low, high);
```

Partition function mainly:
A. Merges arrays
B. Selects pivot position
C. Builds heap
D. Searches element

Answer: ✅ B

100.

```
if(high < low)
   return -1;
```

This base condition belongs to:
A. Linear search
B. Binary search (recursive)
C. Bubble sort
D. Heap sort

Answer: ✅ B

📘 Session 13: Hash Functions & Hash Tables (CCEE)

🟢 EASY LEVEL (1–20)

1. Hashing is mainly used for:

A. Sorting data
B. Fast searching
C. Traversing trees
D. Compression

Answer: ✅ B

---

2. A hash table stores data in the form of:

A. Stack
B. Queue
C. Key–value pairs
D. Tree

Answer: ✅ C

---

3. Hash function maps:

A. Key → Index
B. Index → Key
C. Value → Key
D. Array → Tree

Answer: ✅ A

---

4. Ideal time complexity of search in hashing is:

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

Answer: ✅ C

---

5. Collision occurs when:

A. Table is full
B. Two keys map to same index
C. Hash function fails
D. Table is empty

Answer: ✅ B

---

6. Which is NOT a collision resolution technique?

A. Linear probing
B. Quadratic probing
C. Double hashing
D. Binary search

Answer: ✅ D

---

7. Linear probing resolves collision by:

A. Random jump
B. Searching next available slot sequentially
C. Using second hash function
D. Using linked list

Answer: ✅ B

---

8. Hash table size should preferably be:

A. Even number
B. Prime number
C. Power of 2
D. Multiple of 10

Answer: ✅ B

---

9. Load factor (α) is defined as:

A. table size / keys
B. keys / table size
C. collisions / keys
D. memory / size

Answer: ✅ B

---

10. Hashing is faster than binary search because:

A. No recursion
B. No sorting required
C. Uses divide and conquer
D. Uses tree structure

Answer: ✅ B

---

11. Which collision technique uses another hash function?

A. Linear probing
B. Quadratic probing
C. Double hashing
D. Chaining

Answer: ✅ C

---

12. Hashing with linked list is called:

A. Probing
B. Open addressing
C. Chaining
D. Mapping

Answer: ✅ C

---

13. Worst-case complexity of hashing is:

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

Answer: ✅ C

---

14. Which hash function is simplest?

A. Folding
B. Multiplication
C. Division method
D. Mid-square

Answer: ✅ C

---

15. Hash tables are best for:

A. Ordered data
B. Range queries
C. Exact match queries
D. Traversals

Answer: ✅ C

---

16. In open addressing, collision is resolved:

A. Outside table
B. Inside table
C. Using linked list
D. Using tree

Answer: ✅ B

---

17. Which is NOT a probing technique?

A. Linear
B. Quadratic

C. Double hashing
D. Chaining

Answer: ✅ D

---

18. Hashing improves Fibonacci recursion by:

A. Removing recursion
B. Memoization
C. Iteration
D. Sorting values

Answer: ✅ B

---

19. Hash table deletion is difficult in:

A. Chaining
B. Linear probing
C. Quadratic probing
D. Double hashing

Answer: ✅ B

---

20. A good hash function should:

A. Be slow
B. Cause collisions
C. Distribute keys uniformly
D. Be complex

Answer: ✅ C

🟡 MEDIUM LEVEL (21–40)

21. Linear probing suffers from:

A. Secondary clustering
B. Primary clustering
C. No clustering
D. Rehashing

Answer: ✅ B

---

22. Quadratic probing avoids:

A. Collisions
B. Primary clustering
C. Secondary clustering
D. Rehashing

Answer: ✅ B

---

23. Double hashing reduces:

A. Primary clustering only
B. Secondary clustering only
C. Both primary & secondary clustering
D. Rehashing

Answer: ✅ C

24.

```
index = key % tableSize;
```

This hash function is:
A. Multiplication
B. Folding
C. Division
D. Mid-square

Answer: ✅ C

---

25. If load factor > 1, then:

A. Table is empty
B. More keys than slots
C. No collision
D. Perfect hashing

Answer: ✅ B

---

26. Hashing vs Binary Search:

Binary search requires _____ data.
A. Random
B. Sorted
C. Hashed
D. Indexed

Answer: ✅ B

---

27. In chaining, average search complexity is:

A. O(1 + α)
B. O(n)
C. O(log n)
D. O(α²)

Answer: ✅ A

---

28. Which probing checks indices: h(k)+1², h(k)+2² …?

A. Linear
B. Quadratic
C. Double hashing
D. Chaining

Answer: ✅ B

29.

```
h2(k) = R - (k % R)
```

Used in:
A. Linear probing
B. Quadratic probing
C. Double hashing
D. Division method

Answer: ✅ C

---

30. Deletion in open addressing requires:

A. Removing element
B. Rehashing all elements
C. Marking slot as deleted
D. Clearing table

Answer: ✅ C

---

31. Hashing performs poorly when:

A. Load factor is low
B. Load factor is high
C. Table size is prime
D. Hash function is good

Answer: ✅ B

---

32. Fibonacci optimization using hashing is example of:

A. Greedy
B. Divide & conquer
C. Dynamic programming
D. Backtracking

Answer: ✅ C

---

33. Which collision method is simplest to implement?

A. Double hashing
B. Quadratic probing
C. Linear probing
D. Chaining

Answer: ✅ C

---

34. Hash table rehashing means:

A. Clearing table
B. Increasing table size & reinserting
C. Sorting keys
D. Deleting collisions

Answer: ✅ B

---

35. Hashing is not suitable for:

A. Dictionary
B. Symbol table
C. Range queries
D. Caching

Answer: ✅ C

---

36. In chaining, collision resolution is done using:

A. Array
B. Stack
C. Linked list
D. Tree only

Answer: ✅ C

---

37. If hash table size = 10 and key = 25, index is:

```
h(k) = k % 10
```

A. 2
B. 5
C. 15
D. 0

Answer: ✅ B

---

38. Double hashing uses:

A. One hash function
B. Two hash functions
C. Three hash functions
D. Recursion

Answer: ✅ B

---

39. Best load factor range for hashing:

A. 0 – 0.3
B. 0.7 – 0.9
C. 1 – 2
D. >2

Answer: ✅ B

---

40. Hashing-based search is faster because:

A. No comparisons
B. Direct address calculation
C. Tree traversal
D. Recursion

Answer: ✅ B

🔴 HARD LEVEL (41–60)

41. Primary clustering occurs in:

A. Chaining
B. Quadratic probing
C. Linear probing
D. Double hashing

Answer: ✅ C

---

42. Secondary clustering occurs in:

A. Linear probing
B. Quadratic probing
C. Double hashing
D. Chaining

Answer: ✅ B

---

43. Worst-case of chaining occurs when:

A. Keys distributed uniformly
B. Hash function poor

C. Load factor small
D. Table is empty

Answer: ✅ B

44.

```
while(table[index] != EMPTY)
   index = (index + 1) % size;
```

Which probing?
A. Quadratic
B. Double hashing
C. Linear probing
D. Chaining

Answer: ✅ C

---

45. Which hashing technique is least affected by clustering?

A. Linear probing
B. Quadratic probing
C. Double hashing
D. Chaining

Answer: ✅ C

---

46. Hash table performance mainly depends on:

A. Sorting algorithm
B. Hash function quality
C. Compiler
D. Data type

Answer: ✅ B

---

47. Why prime table size is preferred?

A. Saves memory
B. Reduces collisions
C. Faster computation
D. Required by compiler

Answer: ✅ B

---

48. In Fibonacci recursion optimization, hash table stores:

A. Input values
B. Output results
C. Stack frames
D. Loop counters

Answer: ✅ B

---

49. Which operation is most expensive in hashing?

A. Search (average)
B. Insert (average)
C. Delete (open addressing)
D. Hash calculation

Answer: ✅ C

---

50. If all keys collide into one slot, search complexity becomes:

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

Answer: ✅ C

---

51. Which probing sequence guarantees visiting all slots?

A. Linear probing
B. Quadratic probing
C. Double hashing (if h2 coprime)
D. Chaining

Answer: ✅ C

---

52. Hashing is preferred over BST when:

A. Ordered traversal needed
B. Range queries needed
C. Fast lookup required
D. Memory is tight

Answer: ✅ C

---

53. Deleting element in chaining requires:

A. Rehashing
B. Linked list deletion
C. Marking deleted
D. Table rebuild

Answer: ✅ B

---

54. Which hash function uses middle digits?

A. Folding
B. Division
C. Mid-square
D. Multiplication

Answer: ✅ C

---

55. Open addressing requires:

A. Extra memory
B. Linked list
C. Empty slots
D. Tree

Answer: ✅ C

---

56. Hash table insertion fails when:

A. Load factor = 0
B. Table is full
C. Hash function is good
D. Keys are small

Answer: ✅ B

---

57. In double hashing, second hash function must:

A. Be same as first
B. Return constant
C. Be non-zero & coprime to table size
D. Return prime only

Answer: ✅ C

---

58. Hashing vs Linear Search:

Hashing average search is:
A. Slower
B. Same
C. Faster
D. Equal in worst case

Answer: ✅ C

---

59. Which collision method needs extra memory?

A. Linear probing
B. Quadratic probing
C. Double hashing
D. Chaining

Answer: ✅ D

---

60. Main limitation of hashing:

A. Slow access
B. Poor worst-case performance
C. Needs sorting
D. Complex traversal

Answer: ✅ B

📘 Sessions 14, 15 & 16: Graphs & Applications (CCEE)

🟢 EASY LEVEL (1–25)

1. A graph is defined as a set of:

A. Nodes only
B. Vertices and edges
C. Trees
D. Arrays

Answer: ✅ B

---

2. Graph is considered a generic data structure because:

A. It is simple
B. It can represent many real-world problems
C. It uses arrays
D. It is linear

Answer: ✅ B

---

3. Nodes in a graph are also called:

A. Leaves
B. Vertices
C. Roots
D. Parents

Answer: ✅ B

---

4. Edges represent:

A. Data
B. Relationships
C. Levels
D. Distance only

Answer: ✅ B

---

5. A graph with direction on edges is:

A. Undirected graph
B. Directed graph
C. Weighted graph
D. Cyclic graph

Answer: ✅ B

---

6. A graph with weights on edges is called:

A. Directed graph
B. Simple graph
C. Weighted graph
D. Tree

Answer: ✅ C

---

7. Maximum edges in a simple undirected graph with n vertices:

A. n
B. $n-1$
C. $n(n-1)/2$
D. n^2

Answer: ✅ C

---

8. Which graph has no cycles?

A. Cyclic graph
B. Directed graph
C. Acyclic graph
D. Weighted graph

Answer: ✅ C

---

9. Graph traversal means:

A. Sorting vertices
B. Visiting all vertices

C. Finding shortest path
D. Removing cycles

Answer: ✅ B

---

10. BFS traversal uses:

A. Stack
B. Queue
C. Heap
D. Recursion only

Answer: ✅ B

---

11. DFS traversal uses:

A. Queue
B. Stack / recursion
C. Heap
D. Array only

Answer: ✅ B

---

12. Adjacency matrix representation uses:

A. Linked list
B. 1-D array
C. 2-D array
D. Tree

Answer: ✅ C

---

13. Adjacency list representation uses:

A. Matrix
B. Array of linked lists
C. Stack
D. Queue

Answer: ✅ B

---

14. BFS is also known as:

A. Level order traversal
B. Depth traversal
C. Preorder
D. Inorder

Answer: ✅ A

---

15. DFS explores:

A. Level by level
B. Nearest vertex
C. Depth first
D. Randomly

Answer: ✅ C

---

16. A tree is a special type of:

A. Graph
B. Stack
C. Queue
D. Heap

Answer: ✅ A

---

17. A connected graph has:

A. Cycles
B. All vertices reachable
C. Directed edges
D. Weights

Answer: ✅ B

---

18. Degree of a vertex is:

A. Weight
B. Number of edges incident
C. Distance
D. Level

Answer: ✅ B

---

19. Which structure is better for dense graphs?

A. Adjacency list
B. Adjacency matrix
C. Stack
D. Queue

Answer: ✅ B

---

20. Which structure is better for sparse graphs?

A. Adjacency matrix
B. Adjacency list
C. Heap
D. Tree

Answer: ✅ B

21–25

✔️ Easy questions on graph terminology, basics, representations

---

🟡 MEDIUM LEVEL (26–50)

26. Time complexity of BFS using adjacency list:

A. O(V)
B. O(E)
C. O(V + E)
D. O(V^2)

Answer: ✅ C

---

27. Time complexity of DFS using adjacency matrix:

A. O(V)
B. O(E)

C. $O(V^2)$
D. O(log V)

Answer: ✅ C

---

28. BFS is preferred over DFS when:

A. Memory is low
B. Shortest path in unweighted graph is needed
C. Graph is deep
D. Graph is cyclic

Answer: ✅ B

---

29. DFS is preferred when:

A. Level order needed
B. Backtracking is required
C. Shortest path needed
D. Graph is dense

Answer: ✅ B

---

30. Which graph representation consumes more memory?

A. Adjacency list
B. Adjacency matrix
C. Stack
D. Queue

Answer: ✅ B

---

31. A complete graph with n vertices has edges:

A. n
B. $n-1$
C. $n(n-1)/2$
D. n^2

Answer: ✅ C

---

32. Shortest path in weighted graph (non-negative weights) is found by:

A. BFS
B. DFS
C. Dijkstra
D. Floyd-Warshall

Answer: ✅ C

---

33. Floyd-Warshall algorithm finds:

A. Single-source shortest path
B. Minimum spanning tree
C. All-pairs shortest paths
D. Cycle detection

Answer: ✅ C

---

34. Time complexity of Floyd-Warshall:

A. $O(V^2)$
B. $O(E \log V)$
C. $O(V^3)$
D. $O(V + E)$

Answer: ✅ C

---

35. Dijkstra's algorithm fails when graph has:

A. Positive weights
B. Zero weights
C. Negative weight edges
D. Cycles

Answer: ✅ C

---

36. A spanning tree of a graph:

A. Has all edges
B. Has all vertices and no cycles
C. Has cycles
D. Is disconnected

Answer: ✅ B

---

37. Minimum spanning tree minimizes:

A. Number of vertices
B. Total edge weight
C. Height
D. Degree

Answer: ✅ B

---

38. Prim’s algorithm grows MST by:

A. Sorting edges
B. Choosing minimum edge from tree
C. DFS
D. BFS

Answer: ✅ B

---

39. Kruskal’s algorithm is based on:

A. DFS
B. BFS
C. Greedy approach
D. Dynamic programming

Answer: ✅ C

---

40. Kruskal’s algorithm uses:

A. Stack
B. Queue
C. Union-Find
D. Heap only

Answer: ✅ C

41–50

✔️ Medium questions on complexity, BFS vs DFS, shortest paths, MST

🔴 HARD LEVEL (51–70)

51. BFS guarantees shortest path in:

A. Weighted graph
B. Unweighted graph
C. Directed graph only
D. Cyclic graph

Answer: ✅ B

---

52. Time complexity of Dijkstra (using adjacency matrix):

A. $O(V^2)$
B. O(E log V)
C. O(V + E)
D. O(log V)

Answer: ✅ A

---

53. Which MST algorithm works better for dense graphs?

A. Kruskal
B. Prim (matrix)
C. DFS
D. BFS

Answer: ✅ B

---

54. Which MST algorithm sorts edges first?

A. Prim
B. DFS
C. Kruskal
D. Dijkstra

Answer: ✅ C

---

55. Graph cycle detection in DFS uses:

A. Stack
B. Visited + recursion stack
C. Queue
D. Heap

Answer: ✅ B

---

56. Directed Acyclic Graph (DAG) is used in:

A. Shortest path with negative cycles
B. Scheduling problems
C. MST
D. Sorting numbers

Answer: ✅ B

---

57. Floyd-Warshall can handle:

A. Negative edges only
B. Negative cycles
C. Negative edges (no negative cycles)
D. Only positive edges

Answer: ✅ C

---

58. Graph with V vertices and V−1 edges is:

A. Cyclic
B. Complete
C. Tree
D. Directed

Answer: ✅ C

---

59. Space complexity of adjacency matrix:

A. O(V + E)
B. $O(V^2)$
C. O(E)
D. O(V)

Answer: ✅ B

---

60. Space complexity of adjacency list:

A. $O(V^2)$
B. $O(E^2)$
C. $O(V + E)$
D. $O(V)$

Answer: ✅ C

61–70

✔️ Hard traps on algorithms, complexity, limitations

💻 SNIPPET-BASED MCQs (71–100)

71.

```
for(i=0;i<V;i++)
  for(j=0;j<V;j++)
     adj[i][j] = 0;
```

This initializes:
A. Adjacency list
B. Adjacency matrix
C. Stack
D. Queue

Answer: ✅ B

72.

```
queue.push(src);
visited[src] = 1;
```

Used in:
A. DFS
B. BFS
C. Dijkstra
D. Kruskal

Answer: ✅ B

73.

```
stack.push(v);
```

Used in:
A. BFS
B. DFS
C. Prim
D. Floyd

Answer: ✅ B

74.

```
if(!visited[i])
   dfs(i);
```

This ensures:
A. Sorting
B. Full graph traversal
C. Shortest path
D. Cycle creation

Answer: ✅ B

75.

```
dist[i] = INT_MAX;
dist[src] = 0;
```

Initialization for:
A. BFS
B. DFS
C. Dijkstra
D. Prim

Answer: ✅ C

76.

```
for(k=0;k<V;k++)
 for(i=0;i<V;i++)
  for(j=0;j<V;j++)
```

This loop structure belongs to:
A. Dijkstra
B. BFS
C. DFS
D. Floyd-Warshall

Answer: ✅ D

77.

```
if(find(u) != find(v))
   union(u, v);
```

Used in:
A. Prim
B. DFS
C. Kruskal
D. BFS

Answer: ✅ C

78.

```
minKey(key, mstSet);
```

Used in:
A. Kruskal
B. Prim
C. Dijkstra
D. BFS

Answer: ✅ B

79.

```
for(v=0; v<V; v++)
  if(graph[u][v] && !mstSet[v])
```

Belongs to:
A. DFS
B. BFS
C. Prim
D. Floyd

Answer: ✅ C

80.

```
while(!queue.empty())
```

Used in:
A. DFS
B. BFS
C. Kruskal
D. Prim

Answer: ✅ B

81.

```
void dfs(int v){
  visited[v] = 1;
  for(int i=0;i<V;i++){
    if(adj[v][i] && !visited[i])
       dfs(i);
  }
}
```

This DFS implementation is based on:
A. Adjacency list
B. Adjacency matrix
C. Edge list
D. Stack only

Answer: ✅ B

82.

```
void bfs(int src){
  queue<int> q;
  q.push(src);
  visited[src] = 1;
```

```
}
```

Which data structure is essential for this traversal?
A. Stack
B. Queue
C. Heap
D. Set

Answer: ✅ B

83.

```
if(adj[u][v] != 0)
```

In adjacency matrix, this condition checks:
A. Vertex existence
B. Edge existence between u and v
C. Weight of vertex
D. Degree of graph

Answer: ✅ B

84.

```
for(int i=0;i<V;i++)
   visited[i] = 0;
```

Purpose of this code:
A. Reset graph
B. Initialize visited array
C. Remove edges
D. Create graph

Answer: ✅ B

85.

```
if(!visited[v])
   dfs(v);
```

This ensures:
A. Cycle creation

B. Vertex revisiting
C. Each vertex visited once
D. Shortest path

Answer: ✅ C

86.

```
dist[v] = dist[u] + weight;
```

This operation is known as:
A. Traversal
B. Relaxation
C. Backtracking
D. Heapify

Answer: ✅ B

87.

```
if(dist[u] + w < dist[v])
   dist[v] = dist[u] + w;
```

Used in:
A. BFS
B. DFS
C. Dijkstra
D. Prim

Answer: ✅ C

88.

```
int parent[V];
parent[src] = -1;
```

This array is commonly used to:
A. Store weights
B. Track shortest path tree
C. Detect cycles
D. Count vertices

Answer: ✅ B

89.

```
for(i=0;i<V;i++)
  mstSet[i] = false;
```

This initialization is required for:
A. BFS
B. DFS
C. Prim’s algorithm
D. Floyd-Warshall

Answer: ✅ C

90.

```
key[src] = 0;
parent[src] = -1;
```

This initialization belongs to:
A. Kruskal
B. Prim
C. Dijkstra
D. DFS

Answer: ✅ B

91.

```
sort(edges.begin(), edges.end());
```

This step is essential in:
A. Prim’s algorithm
B. Kruskal’s algorithm
C. Dijkstra’s algorithm
D. BFS

Answer: ✅ B

92.

```
if(find(u) == find(v))
```

```
   continue;
```

This condition avoids:
A. Disconnected graph
B. Negative cycles
C. Cycles in MST
D. Infinite loop

Answer: ✅ C

93.

```
for(int k=0;k<V;k++)
```

In Floyd–Warshall, variable `k` represents:
A. Source vertex
B. Destination vertex
C. Intermediate vertex
D. Edge weight

Answer: ✅ C

94.

```
if(dist[i][k] + dist[k][j] < dist[i][j])
   dist[i][j] = dist[i][k] + dist[k][j];
```

This is core logic of:
A. Dijkstra
B. BFS
C. DFS
D. Floyd-Warshall

Answer: ✅ D

95.

```
if(graph[u][v] && dist[u] != INT_MAX)
```

Why is `dist[u] != INT_MAX` checked?
A. Avoid overflow
B. Improve speed

C. Check adjacency
D. Mark visited

Answer: ✅ A

96.

```
vector<int> adj[V];
```

This declares:
A. Adjacency matrix
B. Edge list
C. Adjacency list
D. Queue

Answer: ✅ C

97.

```
adj[u].push_back(v);
adj[v].push_back(u);
```

This represents:
A. Directed graph
B. Weighted graph
C. Undirected graph
D. Tree

Answer: ✅ C

98.

```
if(!visited[i])
   bfs(i);
```

This helps in:
A. Sorting graph
B. Traversing disconnected graph
C. Finding shortest path
D. Detecting cycles only

Answer: ✅ B

99.

```
int degree = adj[v].size();
```

This calculates:
A. Weight of vertex
B. Degree of vertex
C. Number of paths
D. Tree height

Answer: ✅ B

100.

```
visited[v] = 0;
```

If this line is used after DFS call, it enables:
A. Normal traversal
B. BFS
C. Backtracking / path exploration
D. MST construction

Answer: ✅ C

📘 Sessions 17 & 18: Algorithm Design Techniques (CCEE)

🟢 EASY LEVEL (1–20)

1. An algorithm is:

A. Programming language
B. Step-by-step procedure to solve a problem
C. Compiler
D. Data structure

Answer: ✅ B

---

2. Which is NOT a class of algorithms?

A. Greedy
B. Divide and Conquer
C. Dynamic Programming
D. Compilation

Answer: ✅ D

---

3. Worst-case analysis tells us:

A. Best performance
B. Average performance
C. Maximum time taken
D. Minimum time taken

Answer: ✅ C

---

4. Best-case complexity means:

A. Minimum input size
B. Minimum execution time
C. Random execution
D. Maximum time

Answer: ✅ B

---

5. Average-case complexity considers:

A. Worst inputs only
B. Best inputs only
C. All possible inputs
D. Only sorted inputs

Answer: ✅ C

---

6. Which algorithm technique breaks problem into subproblems?

A. Greedy
B. Divide and Conquer
C. Brute Force
D. Stochastic

Answer: ✅ B

---

7. Greedy algorithms make decisions based on:

A. Future results
B. Global optimum
C. Local optimum
D. Random choice

Answer: ✅ C

---

8. Dynamic Programming mainly avoids:

A. Recursion
B. Loops
C. Recomputing subproblems
D. Memory usage

Answer: ✅ C

---

9. Brute force approach is:

A. Optimal
B. Complex
C. Straightforward & exhaustive
D. Always efficient

Answer: ✅ C

---

10. Backtracking is useful when:

A. All solutions required
B. Only one solution
C. No constraints
D. Linear problems

Answer: ✅ A

---

11. Branch and Bound is an improvement over:

A. Greedy
B. Backtracking

C. Divide & Conquer
D. Brute force

Answer: ✅ B

---

12. Stochastic algorithms involve:

A. Deterministic steps
B. Randomness
C. Sorting
D. Recursion

Answer: ✅ B

---

13. Time complexity measures:

A. Memory usage
B. Execution time growth
C. Lines of code
D. CPU speed

Answer: ✅ B

---

14. Space complexity measures:

A. Execution speed
B. Code size
C. Memory used
D. Number of operations

Answer: ✅ C

---

15. Asymptotic analysis ignores:

A. Constants
B. Input size
C. Growth rate
D. Complexity

Answer: ✅ A

---

16. Which notation represents upper bound?

A. Ω
B. Θ
C. O
D. σ

Answer: ✅ C

---

17. Which notation represents lower bound?

A. O
B. Ω
C. Θ
D. ∞

Answer: ✅ B

---

18. Θ notation represents:

A. Upper bound
B. Lower bound
C. Tight bound
D. Worst case only

Answer: ✅ C

---

19. Efficient algorithm means:

A. Less readable
B. Optimal time/space
C. More code
D. Uses recursion

Answer: ✅ B

---

20. Choosing an algorithm depends mainly on:

A. Syntax
B. Compiler
C. Nature of problem & data
D. Programming language

Answer: ✅ C

🟡 MEDIUM LEVEL (21–40)

21. Merge sort follows:

A. Greedy
B. Divide & Conquer
C. DP
D. Brute force

Answer: ✅ B

---

22. Dijkstra's algorithm is:

A. DP
B. Greedy
C. Backtracking
D. Brute force

Answer: ✅ B

---

23. Fibonacci with memoization is:

A. Greedy
B. Divide & Conquer
C. Dynamic Programming
D. Brute force

Answer: ✅ C

---

24. N-Queens problem uses:

A. DP
B. Greedy
C. Backtracking
D. Branch & Bound

Answer: ✅ C

---

25. Traveling Salesman (exact solution) uses:

A. Greedy
B. DP
C. Branch & Bound
D. BFS

Answer: ✅ C

---

26. Which algorithm design uses recursion heavily?

A. Greedy
B. Divide & Conquer
C. Brute force
D. Stochastic

Answer: ✅ B

---

27. Which technique guarantees optimal solution?

A. Greedy (always)
B. DP
C. Random
D. Brute force (always optimal but costly)

Answer: ✅ D

---

28. Greedy algorithm may fail because:

A. Too slow
B. Uses recursion
C. Local optimum ≠ global optimum
D. Uses extra space

Answer: ✅ C

---

29. Which problem is best solved using DP?

A. Binary search
B. Longest Common Subsequence
C. Linear search
D. Sorting array

Answer: ✅ B

---

30. Backtracking improves performance by:

A. Ignoring constraints
B. Pruning invalid paths
C. Random choice
D. Using heaps

Answer: ✅ B

---

31. Branch & Bound differs from backtracking by:

A. Using recursion
B. Using bounds to prune
C. Finding all solutions
D. No pruning

Answer: ✅ B

---

32. Worst-case of quick sort:

A. O(n log n)
B. $O(n^2)$
C. O(n)
D. O(log n)

Answer: ✅ B

---

33. Which algorithm is not deterministic?

A. Greedy
B. DP
C. Stochastic
D. Divide & Conquer

Answer: ✅ C

---

34. Algorithm analysis is independent of:

A. Input size
B. Hardware
C. Growth rate
D. Complexity

Answer: ✅ B

---

35. Which is NOT part of complexity analysis?

A. Time
B. Space
C. Correctness
D. Growth rate

Answer: ✅ C

---

36. Which algorithm explores all possibilities?

A. Greedy
B. DP
C. Brute force
D. Divide & Conquer

Answer: ✅ C

---

37. Greedy algorithm example:

A. Merge sort
B. Kruskal’s algorithm
C. Floyd-Warshall
D. Binary search

Answer: ✅ B

---

38. DP requires:

A. Random choice
B. Optimal substructure
C. Linear data
D. No recursion

Answer: ✅ B

---

39. Backtracking is commonly used in:

A. Graph MST
B. Scheduling
C. Constraint satisfaction problems
D. Searching

Answer: ✅ C

---

40. Choosing wrong algorithm mainly affects:

A. Syntax
B. Performance
C. Compilation
D. Output correctness

Answer: ✅ B

🔴 HARD LEVEL (41–60)

41. Which technique trades space for time?

A. Greedy
B. Backtracking
C. Dynamic Programming
D. Brute force

Answer: ✅ C

---

42. Which algorithm always gives optimal solution but is inefficient?

A. Greedy
B. DP
C. Brute force
D. Stochastic

Answer: ✅ C

---

43. Worst-case complexity of backtracking is:

A. Polynomial
B. Exponential
C. Linear
D. Logarithmic

Answer: ✅ B

---

44. Which design technique is best for optimization problems?

A. DP
B. Brute force
C. Linear search
D. Traversal

Answer: ✅ A

---

45. Branch & Bound reduces:

A. Correctness
B. Search space
C. Recursion
D. Memory

Answer: ✅ B

---

46. Which algorithm class uses probability?

A. Greedy
B. DP
C. Stochastic
D. Divide & Conquer

Answer: ✅ C

---

47. Which complexity dominates?

A. O(n)
B. O(log n)
C. O(n log n)
D. $O(n^2)$

Answer: ✅ D

---

48. DP differs from Divide & Conquer by:

A. Recursion
B. Overlapping subproblems
C. Problem division
D. Sorting

Answer: ✅ B

---

49. Which algorithm is fastest on nearly sorted data?

A. Merge sort
B. Heap sort
C. Insertion sort
D. Quick sort

Answer: ✅ C

---

50. Algorithm efficiency improves by:

A. Increasing memory
B. Reducing complexity
C. Using recursion
D. Writing more code

Answer: ✅ B

---

51. Greedy algorithms fail in:

A. MST
B. Shortest path (with negative edges)
C. Scheduling
D. Coin change (canonical only)

Answer: ✅ B

---

52. DP is not suitable when:

A. Subproblems overlap
B. Optimal substructure exists

C. Memory is very limited
D. Recursive solution exists

Answer: ✅ C

---

53. Which technique is used in AI search?

A. Branch & Bound
B. Brute force
C. Binary search
D. Sorting

Answer: ✅ A

---

54. Which algorithm design is easiest to implement?

A. DP
B. Greedy
C. Backtracking
D. Branch & Bound

Answer: ✅ B

---

55. Which complexity grows fastest?

A. $O(n \log n)$
B. $O(n^2)$
C. $O(2^n)$
D. $O(n^3)$

Answer: ✅ C

---

56. Case study comparison helps in:

A. Syntax learning
B. Algorithm selection
C. Debugging
D. Compilation

Answer: ✅ B

---

57. Which technique solves problems incrementally?

A. Divide & Conquer
B. Greedy
C. DP
D. Backtracking

Answer: ✅ B

---

58. Worst-case analysis is important because:

A. Rare
B. Guarantees upper bound
C. Average case is enough
D. Saves memory

Answer: ✅ B

---

59. Algorithm design focuses on:

A. Code syntax
B. Problem-solving strategy
C. Language features
D. Compilation

Answer: ✅ B

---

60. Application of data structures in algorithm design helps to:

A. Reduce complexity
B. Increase code
C. Slow execution
D. Add overhead

Answer: ✅ A

💻 SNIPPET-BASED MCQs (61–80)

61.

```
T(n) = 2T(n/2) + n
```

Which technique?
A. Greedy
B. Divide & Conquer
C. DP
D. Brute force

Answer: ✅ B

62.

```
if(choice_is_feasible)
   take_choice();
```

This logic represents:
A. DP
B. Greedy
C. Backtracking
D. Brute force

Answer: ✅ B

63.

```
if(dp[n] != -1)
   return dp[n];
```

Used in:
A. Greedy
B. DP (Memoization)
C. Brute force
D. Backtracking

Answer: ✅ B

64.

```
for(all possibilities)
   check_solution();
```

Which approach?
A. Greedy

B. DP
C. Brute force
D. Branch & Bound

Answer: ✅ C

65.

```
if(cost > bound)
   prune();
```

Which technique?
A. Backtracking
B. Branch & Bound
C. Greedy
D. DP

Answer: ✅ B

66.

```
sort(edges);
pick_min_edge();
```

This belongs to:
A. DP
B. Greedy
C. Divide & Conquer
D. Brute force

Answer: ✅ B

67.

```
solve(left);
solve(right);
combine();
```

Which paradigm?
A. Greedy
B. DP

C. Divide & Conquer
D. Backtracking

Answer: ✅ C

68.

```
for(i=1;i<=n;i++)
   dp[i] = min(dp[i-1], dp[i-2]);
```

Which approach?
A. Greedy
B. DP
C. Backtracking
D. Brute force

Answer: ✅ B

69.

```
random_choice();
```

Which algorithm type?
A. Greedy
B. DP
C. Stochastic
D. Divide & Conquer

Answer: ✅ C

70.

```
if(is_safe)
   recurse_next();
```

This is typical of:
A. DP
B. Backtracking
C. Greedy
D. Divide & Conquer

Answer: ✅ B

71.

```
T(n) = T(n-1) + n
```

Time complexity?
A. O(n)
B. O(n log n)
C. O(n^2)
D. O(2^n)

Answer: ✅ C

72.

```
T(n) = T(n/2) + 1
```

Complexity?
A. O(n)
B. O(log n)
C. O(n log n)
D. O(n^2)

Answer: ✅ B

73.

```
T(n) = T(n-1) + T(n-2)
```

Typical of:
A. DP optimized
B. Greedy
C. Recursive Fibonacci
D. Divide & Conquer

Answer: ✅ C

74.

```
if(current_cost >= best_cost)
   return;
```

Used in:
A. Backtracking
B. Greedy
C. Branch & Bound
D. DP

Answer: ✅ C

75.

```
for(i=0;i<n;i++)
 for(j=0;j<n;j++)
```

Time complexity?
A. O(n)
B. O(n log n)
C. O(n^2)
D. O(2^n)

Answer: ✅ C

76.

```
solve(n-1);
solve(n-1);
```

Time complexity?
A. O(n)
B. O(log n)
C. O(2^n)
D. O(n^2)

Answer: ✅ C

77.

```
best = min(best, current);
```

Indicates:
A. DP
B. Greedy
C. Branch & Bound
D. Sorting

Answer: ✅ C

78.

```
choose();
explore();
unchoose();
```

This is pattern of:
A. DP
B. Greedy
C. Backtracking
D. Divide & Conquer

Answer: ✅ C

79.

```
while(problem_not_solved)
   make_best_choice();
```

Represents:
A. DP
B. Greedy
C. Backtracking
D. Brute force

Answer: ✅ B

80.

```
use_priority_queue();
```

Commonly used in:
A. Greedy algorithms
B. Brute force
C. DP only
D. Backtracking

Answer: ✅ A