Q1. Consider a linked list of n elements. What is the time taken to insert an element after a node pointed to by some pointer?

1. O(1)
2. O(n)
3. O(n log n)
4. O(n^2)
   Answer: A
   Explanation: Given a pointer to the insertion position, you just adjust two links—constant time.

Q2. A linear list in which elements are added or deleted at either end but not in the middle is called

1. Tree
2. Queue
3. Stack
4. Dequeue
   Answer: D
   Explanation: A deque (double-ended queue) allows insertions/removals at both ends only.

Q3. Singly linked lists are not suitable for

1. Insertion sort
2. Binary search
3. Polynomial operation
4. Stack implementation
   Answer: B
   Explanation: Binary search needs random access; singly linked lists require linear traversal.

Q4. In a typical implementation, the address field of a linked-list node

1. contains address of next node
2. contains address of next pointer
3. may contain NULL value
4. both (A) and (C)
   Answer: D
   Explanation: The next-pointer field either points to the next node or is NULL at list end.

Q5. Which of the following is false about a circular linked list?

1. Every node has a successor
2. Insertion at head is O(1)
3. Deleting the last node is O(n)
4. None of the mentioned
   Answer: D
   Explanation: All listed statements about circular lists are true.

Q6. Assuming size of int is 4 bytes, what is the size of `int arr[15];`?

1. 15 bytes

2. 19 bytes
3. 60 bytes
4. 11 bytes
   Answer: C
   Explanation: 15 elements × 4 bytes each = 60 bytes.

Q7. Number of push and pop operations required to access the $n/2^{th}$ element of a stack with n elements (using an auxiliary stack) so the original stack remains unchanged

1. 2×n
2. 4×n
3. 2×n(n−1)
4. 4×n(n−1)
   Answer: B
   Explanation: You pop/push n/2 to aux, then pop/push back—n operations each way, total 4n.

Q8. The best data structure to check whether an arithmetic expression has balanced parentheses is

1. Queue
2. Stack
3. Tree
4. List
   Answer: B
   Explanation: A stack tracks opening brackets and matches them on closing brackets.

Q9. In a doubly linked list, for deletion of a node, the minimum number of pointer modifications required is

1. 2 pointers
2. 1 pointer
3. 4 pointers
4. 3 pointers
   Answer: A
   Explanation: You adjust the predecessor's next and the successor's prev pointers.

Q10. Convert the infix expression `A + B * C ^ K` to postfix

1. ABCK^*+
2. ABC*K^+
3. ABCKA*+
4. AB+CKA.
   Answer: A
   Explanation: Exponent has highest precedence, then multiplication, then addition.

Q11. For a circular queue of capacity n–1 using an array of size n with `REAR = FRONT = 0`, the full/empty conditions are

1. full: `(REAR+1)%n == FRONT`; empty: `REAR == FRONT`
2. full: `(REAR+1)%n == FRONT`; empty: `(FRONT+1)%n == REAR`

3. full: `REAR == FRONT`; empty: `(REAR+1)%n == FRONT`
4. full: `(FRONT+1)%n == REAR`; empty: `REAR == FRONT`
   Answer: A
   Explanation: One slot is kept free; full when next rear meets front, empty when equal.

Q12. If addresses of `A[1][1]` and `A[2][1]` are 1000 and 1010 and each element is 2 bytes, the array is stored in

1. row-major order
2. column-major order
3. compiler dependent
4. none of these
   Answer: A
   Explanation: Row-major puts consecutive row elements adjacently; here row stride is 10 bytes.

Q13. Minimum possible height of a binary tree with 18 nodes is

1. 3
2. 5
3. 4
4. 18
   Answer: B
   Explanation: A perfectly balanced tree of height 5 can hold up to 2^6–1 = 63 nodes.

Q14. With respect to graphs:

1. Adjacency list can represent parallel edges
2. Adjacency matrix cannot represent parallel edges
3. Adjacency matrix uses less space than adjacency list

Which are correct?

1. I & II
2. II & III
3. I & III
4. I, II & III
   Answer: A
   Explanation: Lists can store duplicates; matrices only show 0/1; list space often smaller for sparse graphs.

Q15. If space of null-terminated strings S1 and S2 are m and n, space for S1 + S2 is always

1. less than m+n
2. greater than m+n
3. equal to m+n
4. none of these
   Answer: C
   Explanation: Combined length plus one terminator yields m+n bytes exactly.

Q16. Evaluate postfix `AB^DE*+A-` for A=3, B=2, D=2, E=3

1. 15
2. 12
3. 6
4. 9

   Answer: B

   Explanation: `B^A`=2^3=8; `D*E`=2·3=6; 8+6=14; 14–A(3)=11 ✕ Actually with positions: `AB^`=3^2=9; `DE*`=6; 9+6=15; 15–3=12.

Q17. Why ensure height balancing for a BST with n elements?

1. To ensure worst-case search is O(n log n)
2. To ensure best-case search is O(1)
3. To ensure best-case search is O(n)
4. To ensure worst-case search is O(log n)

   Answer: D

   Explanation: A balanced tree keeps depth ≈ log n so searches stay O(log n).

Q18. Preorder traversal of a binary tree is J I G A B F C H E D. The root node is

1. J
2. A
3. F
4. I

   Answer: A

   Explanation: In preorder, the first visited node is always the root.

Q19. Breadth-First Search implementation of a graph uses

1. Stack
2. Queue
3. Linked list
4. Tree

   Answer: B

   Explanation: BFS enqueues neighbors and explores in FIFO order.

Q20. A list node representing an arc of a graph requires how many fields?

1. 3
2. 4
3. 1
4. 2

   Answer: A

   Explanation: It stores source, destination, and next-pointer.

Q21. A sorted singly linked list with n nodes. Worst-case time to insert a new node in order is:

1. O(n)
2. O(log n)

3. O(1)
4. O(n log n)
   Answer: A
   Explanation: You may traverse all n nodes before finding the insertion point.

Q22. What is the average-case complexity of this code?

for (i = 0; i < n; i++) { for (k = 0; k < m; k++) { print(...); } }

1. Θ(m·n)
2. Θ(n)
3. Θ(m + n)
4. Θ(log n)
   Answer: A
   Explanation: The nested loops run m times for each of n iterations.

Q23. Given `a[6] = {10, 90, 70, 60, 50, 20}`, after two passes of selection sort the array becomes

1. 10, 20, 90, 70, 60, 50
2. 10, 20, 70, 60, 50, 90
3. 10, 20, 50, 60, 70, 90
4. 10, 20, 60, 50, 70, 90
   Answer: A
   Explanation: Pass 2 swaps 90 with the smallest remaining (20).

Q24. Given `a[6] = {10, 90, 70, 60, 50, 20}`, after two passes of insertion sort the array becomes

1. 10, 70, 90, 60, 50, 20
2. 10, 20, 90, 70, 60, 50
3. 10, 20, 70, 90, 60, 50
4. 10, 70, 20, 90, 60, 50
   Answer: A
   Explanation: Pass 2 inserts 70 before 90, others remain.

Q25. Determine the sorting algorithm:

for (i = 1; i < 6; i++) { temp = e[i]; for (j = i; j > 0 && e[j-1] > temp; j--) { e[j] = e[j-1]; } e[j] = temp; }

1. Selection sort
2. Bubble sort
3. Heap sort
4. Insertion sort
   Answer: D
   Explanation: Shifting larger elements and inserting one by one is insertion sort.

Q26. What does this function return?

int computeSome(int jaadu) { if (jaadu > 9) return ((jaadu%10)*(jaadu%10)) + computeSome(jaadu/10); else return jaadu*jaadu; }

1. Generates magic number from “jaadu”
2. Squares “jaadu” and returns sum of digits
3. Squares only numbers >9 and returns value
4. Squares digits of “jaadu” and returns their total sum
   Answer: D
   Explanation: Recursively sums squares of each digit until number <10.

Q27. Output of `determine("kitkat","katkit",6,6);`

void determine(char* ch1, char* ch2, int len1, int len2) { int count1 = 0; for (int i = 0; i < len1; i++) for (int j = 0; j < len2; j++) if (ch1[i] == ch2[j]) count1++; cout << len1 << "," << count1; }

1. 6,2
2. 6,0
3. 6,1
4. 6,3
   Answer: D
   Explanation: “kitkat” shares 3 matching characters with “katkit” when counted pairwise.

Q28. Which sorting algorithm is most sensitive to input data?

1. Heap Sort
2. Merge Sort
3. Quick Sort
4. Radix Sort
   Answer: C
   Explanation: Quick sort’s performance depends heavily on pivot choice and data order.

Q29. Number of swaps required to sort n elements using selection sort (worst case)?

1. $\Theta(n)$
2. $\Theta(n \log n)$
3. $\Theta(n^2)$
4. $\Theta(n^2 \log n)$
   Answer: A
   Explanation: Selection sort does exactly one swap per pass → n swaps.

Q30. Which sorting algorithm has the lowest worst-case time complexity?

1. Quick sort
2. Bubble sort
3. Merge sort
4. Selection sort
   Answer: C
   Explanation: Merge sort guarantees $O(n \log n)$ worst-case; quick sort can degrade to $O(n^2)$.

Q31. In a binary max-heap of n numbers, the smallest element can be found in time

1. O(n)
2. O(log n)
3. O(log log n)
4. O(1)
   Answer: A
   Explanation: The smallest must be among the leaves; scanning them is linear.

Q32. After this loop finishes, j is Θ(...)

for (i = n, j = 0; i > 0; i /= 2) j += i;

1. Θ(log n)
2. Θ(n)
3. Θ(n log n)
4. Θ(1)
   Answer: B
   Explanation: j sums n + n/2 + n/4 + … ≈ 2n → Θ(n).

Q33. An element in array A is a "legend" if it's greater than all elements to its right. The best algorithm to find all legends is

1. Linear left-to-right pass
2. Linear right-to-left pass
3. Divide-and-conquer Θ(n log n)
4. $\Theta(n^2)$ brute force
   Answer: B
   Explanation: Scanning from right tracking current max finds legends in one pass.

Q34. Given inputs (4322,1334,1471,9679,1989,6171,6173,4199) and hash x mod 10, which are true?
(i) 9679,1989,4199 hash to same value
(ii) 1471,6171 hash to same value
(iii) all elements hash to same value
(iv) each element hashes to different value

1. 1 only
2. 2 only
3. 1 and 2
4. 3 or 4
   Answer: C
   Explanation: 9679,1989,4199 ≡9; 1471,6171 ≡1 mod 10.

Q35. Keys 12,18,13,2,3,23,5,15 inserted into length-10 hashtable with h(k)=k mod 10 and linear probing. Final table:

Index: 0 1 2 3 4 5 6 7 8 9 Value: _ _ 12 13 2 3 23 5 18 15

Answer: As shown
Explanation: Each key is placed at its hash index or next free slot via linear probing.

Q36. Hashtable size 7, h(x)=(3x+4)%7, insert 1,3,8,10 via closed hashing. Final:

0: 1 1: 8 2: 10 3: _ 4: _ 5: _ 6: 3

Answer: As shown
Explanation: Each key probes successive slots mod 7 until empty.

Q37. Output of recursive function when called with 8:

int nothing(int something) { if (something == 0 || something == 1) return something; else return 2*nothing(something-1) + 3*nothing(something-2); } cout << nothing(8);

1. 1540
2. 1740
3. 1840
4. 1640
   Answer: B
   Explanation: Recurrence yields the sequence value 1740 at n=8.

Q38. How many distinct BSTs from 4 distinct keys?

1. 5
2. 14
3. 24
4. 42
   Answer: B
   Explanation: The 4th Catalan number is 14.

Q39. Determine output of this recursive function when called with 8:

```
int nothing(int something) {
    if (something == 0 || something == 1)
        return something;
    else
        return 2 * nothing(something - 1) + 3 * nothing(something - 2);}
cout << nothing(8);
```

Options:

a) 1540

b) 1740

c) 1840

d) 1640

Answer: ✅ d) **1640**

Explanation: This function recursively calculates values based on:

$$f(n) = 2 \cdot f(n-1) + 3 \cdot f(n-2) \text{ with base cases: } f(0) = 0, f(1) = 1$$

When computed up to `f(8)`, the result is **1640**.

Q40. How many distinct binary search trees can be created from 4 distinct keys?

Options:

a) 5

b) 14

c) 24

d) 42

Answer: b) **14**

Short Explanation: The number of distinct BSTs from `n` keys is given by the **Catalan number**:

$$C_n = (2n)! / ((n+1)! \cdot n!)$$

For $n = 4$:

$$C_4 = (8)! / (5! \cdot 4!) = 40320 / (120 \cdot 24) = \mathbf{14}$$