DSA MTE

① Asymptotic Notation
It describes how an algorithm's running time or space grows with ~~time~~ input size n for large n.

* It has three types -
(i) Big O → worst case
(ii) Big Ω → best case
(iii) Big Θ → average case

② Tail Recursion	Head Recursion
Recursive call occurs in the last of statement	Recursive call occurs first.
Eg → ascending order	descending order

Address of A[~~i~~][~~j~~] = B + W * (~~i~~ * n + j)
n → elements

③ Linear Search → Searches for an element linearly → one by one

TC → O(n)

Binary Search → Searches for an element by dividing the array into smaller parts. Checks the elements until it reaches to one element after dividing. TC → O(log n)

(5)

(i) Insertion Sort Algorithm

(ii) Start from the 2nd element [i = 1].

(iii) Compare it with elements on the left.

(iv) Shift all elements greater than it to one position to the right.

(v) Place the element in it correct position.

(vi) Move to the next element and repeat until the whole list is sorted.

(vii) Stop

(6) Sparse Matrix has 0 elements. Storing all elements wastes memory, therefore we store only non-zero elements.
They are represented through linked lists.

(7) Reversing a Singly LL

For reversing, we change the direction of its links so that each node points to its previous node.

Steps
(1) Initialise pointers
(i) prev = NULL
(ii) curr = head
(iii) next = NULL

(2) Traverse the list
(i) Save next node → next = curr → next
(ii) Reverse link → curr → next = prev
(iii) Move forward → prev = curr, curr = next

(3) Finally, set head = prev
TC ≥ O(n) SC = O(1)

(8) (a) Algo → Tower (n, source, auxillary, destination)
if (n == 1)
move disk from source to destination
return
Tower (n-1, source, destination, auxillary)
move disk from source to destination
Tower (n-1, auxillary, source, destination)

Explanation

* Move top n-1 disks from source to auxillary.
* move the largest disk to destination
* move n-1 disks from auxillary to destination.

(b)

	Recursion	Iteration
Memory use	Uses call stack for each call	Uses constant memory.
Speed	Slower due to function call overhead.	Faster, no call overhead.
Code clarity	Easier for divide and conquer problems.	Sometimes more complex.
Termination	Needs base condn.	Uses loop condn.

(3) (a) Algo

```
merge sort (arr, l, r);
if(l < r)
mid = (l+r)/2
merge sort (arr, l, mid)
merge sort (arr, mid+1, r)
merge (arr, l, mid, r)
```

TC = O(n log n) SC = O(n)

* Benefits over Bubble sort
(i) Merge sort is faster
(ii) Works efficiently on large datasets.
(iii) Stable and predictable performance.