DSA MTE

① Asymptotic Notation
It describes how an algorithm's running time or space grows with ~~time~~ input size 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 → $O(\log n)$

(3)

Insertion Sort Algorithm

(i) start

(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 three 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 destinatio
- * 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.

9 (a) Algo

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

TC = O(n logn) SC = O(n)

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

* set-2

1. Time-space trade-off is the concept where sometimes more memory (space) can be used to make a process faster and vice-versa. for eg.→ Storing values in array.

2. Multidimensional, arrays are useful for representing matrix data, images and game boards.

int [][] matrix = {{1,2},{3,4}}

Ans-3 For a 1-D array arr, element arr[i] is at index i.

This formula is important as it allows constant time.

Ans 4
```
int LS(int[]arr, int value){
for(int i=0; i<arr.length; i++){
 if (arr[i]== value)
 { return i;
  }
   return -1;
  }
```
Adv → Simple and works for any array.
Limitation → Slow for large arrays.

Ans-5 Quick sort picks a pivot, positions the array so left are smaller and right are larger and quick sorts each partition

Ans 6 * Direct Recursion → function calls itself directly.

* Indirect Recursion → Function calls another fuction, which then calls original.

```
void direct Rec (int n) {
if (n>0) direct Rec (n-1);
 }
void recA ( int n) { if (n>0)
recB (n-1); }
void rec B (int n) { if (n>0)
rec A(n-1); }
```

A-7 Insertion operation creates new node, adjust previous/next pointers of neighbouring nodes.

Deletion operation removes node, adjust pointers to "skip" deleted node.

8(a)

```
int fibonacci (int n) {
if ( n <= 1) return n;
int a=0, b = 1, c;
for ( int i = 2; i <= n; i++) {
   c = a+b;
   a = b;
   b = c;
  }
  return b;
 }
```

(b) Removal of recursion with an en→ Recursion for Fibonacci can be replaced with above loop, saving stack space.

⑨ Store only non-zero elements as nodes containing row/column/value. Benefits - saves memory compared to standard array for large, mostly zero matrices.

SET-3

① Algorithm efficiency refers to how well an algorithm uses ~~spac~~ resources, especially time and space such as how fast it runs or how much memory it uses. It is measured using time complexity & space complexity

②(i) TC shows how running time increases with input size. (In linear search O(n) as it may scan every element).

(ii) SC describes how much memory is used (eg. an extra array in merge sort uses O(n) space.

③ Index = K * x * y + j * x + i

④ (i) Find the middle of the sorted array

(ii) If the middle is the target, return its index.

(iii) If the target is smaller, repeat search in left half, if larger, then in right half.

(5) Bubble sort repeatedly compare adjacent elements and swap it out of order. After each pass, the largest unsorted element "bubbles" to the end.

```
void bubble sort(int[]arr){
for(int i=0; i<arr.length-1, i++){
for(int j=0; j<arr.length-1;j++){
 if(arr[j] > arr[j+1])
{
 int temp = arr[j];
 arr[j]=arr[j+1];
 arr[j+1] = temp;
 }
 }
 }
```

⑥ The factorial of n:(n!) is

n * (n-1) * ----- * 1.

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

⑧ (a) Discuss trade off b/w recursion and iteration (memory)

* Recursion uses more memory since each call adds to the call stack.
* Iteration uses a fixed amount of memory, which is more efficient.

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

⑨ * Merge Sort divides the array into halves, sorts each, and merges the results using extra memory.
* Used in external sorting (like in disk) large data sets or where stability is required.

SET - 4

① An algo is a finite sequence of steps to solve a problem. Efficiency is important because it determines the time and space complexity needed by algorithm, impacting performance especially for large datasets.

② In row-major order, elements of a row and stored in contiguous memory locations, in column-major order elements of a column are contiguous.
Diagram should show 2D array labelled to indicate access pattern for each scheme.

③ For an n-dimensional array ~~sorted~~ sorted in row-major order.

Address = Base + $\sum_{i=1}^{n}$ (i current * product)

This formula calculates the linear memory location for element.

④ Insertion sort can be more efficient for partially sorted data, while

Selection sort does fewer steps but always performs the same number of comparisons.
Both have $O(n^2)$ time complexity in worst and average cases.

⑤ Merge Sort is a divide-and-conquerer algorithm.
It divides the array into halves. recursively sorts them, and merges the results.

⑥ Recursion can often be replaced with loops for efficiency.

```
int factorial (int n){
int result = 1;
for(int i=1; i<=n; i++){
  result *= i;
  return result;
 }
```