1. Define asymptotic notation and explain its importance in analyzing algorithm efficiency.?
ans:- Asymptotic notations are mathematical tools used to represent the growth rate of an algorithm's running time or space requirement as the input size becomes very large.
- these notations describe the limiting behavior of an algorithm and help compare the performance of different algorithms.

The main notations are:
Big O (O) - describes the upper bound or worst-case performance. Example: $T(n)=3n^2+2n+1 \rightarrow O(n^2)$.
Omega (Ω) - gives the lower bound or best-case performance.
Theta - gives the average or tight bound.

importance:
They allow algorithm comparison independent of system or hardware.
they help predict scalability for large datasets.
they guide in choosing the most efficient algorithm for

implementation.

2. Explain the difference between tail recursion and head recursion with examples.
ans) Head Recursion:
the recursive call is made before any computation.
the function completes all recursive calls first and then performs operations while returning.

Example:
```
void fun(int n){
if(n>0){
fun(n-1);
printf("%d ",n);
    }
}
```
For n=3, the output will be 1 2 3.

Tail Recursion:
the recursive call is made after all computations.
there is no pending operation after the recursive call returns.

Example:
```
void fun(int n){
```

```
if(n>0){
printf("%d ",n);
fun(n-1);
   }
}
```

For n=3, the output will be 3 2 1.

Difference: Tail recursion can be optimized into iteration easily and uses less stack memory compared to head recursion.

3. Derive the index formula for accessing elements in a 2-D array stored in row-major order?
ans) Consider a two-dimensional array A[L1..U1][L2..U2] where each element occupies 'w' bytes and the base address is BA.

The formula for calculating the address of element A[i][j] in row-major order is:
LOC(A[i][j]) = BA + [(i-L1) × (U2 - L2 + 1) + (j - L2)] * w

Explanation:
- in row-major order, all elements of the first row are stored first, followed by the next

rows.
- to reach A[i][j], we skip (i-L1) complete rows and then (j-L2) elements in the current row.

Example: For A[3][4] and each element of 2 bytes, LOC(A[2][3]) = BA + ((2×4)+3)×2.

4. Explain the difference between linear search and binary search with their time complexities.
ans) Linear Search :

Works on both sorted and unsorted arrays.
Sequentially compares each element with the target key.
Best case: O(1) if found at the beginning.
Worst case: O(n) if at the end or not present.
Simple to implement but inefficient for large datasets.

Binary Search:
Works only on sorted arrays.
Compares the key with the middle element, and based on comparison, it discards half of the array each time.
Time complexity is O(log n) for average and worst

case.
Space complexity is O(1) for iterative and O(log n) for recursive version.

Conclusion: Binary search is much faster than linear search for large sorted data because it reduces the search space by half each time.

5. Write the algorithm and explain the working of insertion sort with an example.
ans) Algorithm:

- Start with the second element (index 1) and consider it as the key.
- Compare the key with all elements before it.
- Shift all greater elements one position to the right.
- insert the key in its correct position.
- repeat for all elements until the array is sorted.

Example:
Input array: [5, 3, 4, 1]
Pass 1: Insert 3 → [3, 5, 4, 1]
Pass 2: Insert 4 → [3, 4, 5, 1]
Pass 3: Insert 1 → [1, 3, 4,

5]

Time Complexity: Best case O(n), worst case O(n²).
Space Complexity: O(1).

Advantages: Stable, in-place sorting and efficient for small or nearly sorted arrays.

6. Discuss how sparse matrices are represented and explain any one representation method.?
ans) a sparse matrix is a matrix in which most of the elements are zero.
- Storing all elements, including zeros, wastes memory, so special methods are used to store only non-zero elements.
- Common representation methods are:

triplet representation (array of row, column, value).
Linked list representation.
triplet Method Explanation:

- the first row stores the dimensions of the matrix and the number of non-zero elements.
- Each subsequent row stores three values: row

index, column index, and the non-zero value.

Example:
Matrix
[0 4 0]
[0 0 5]
[7 0 0]
Triplet form:
(3 3 3)
(0 1 4)
(1 2 5)
(2 0 7)
Advantages: Saves space, easy to transpose, and efficient for matrix operations involving mostly zero elements.

7. Explain the process of reversing a singly linked list with an example?
ans) a singly linked list contains nodes where each node has two parts: data and a pointer to the next node.
- to reverse the list, we change the direction of the links so that each node points to its previous node instead of the next

one.

Algorithm:
- initialize three pointers: prev = NULL, curr = head, next = NULL.
- Repeat while curr is not NULL:
a. Store curr->next in next.
b. Change curr->next to prev.
c. Move prev to curr.
d. Move curr to next.
Finally, set head = prev.

Example: Original list: 10 → 20 → 30 → NULL
After reversal: 30 → 20 → 10 → NULL
Complexity: Time O(n), Space O(1).
Application: Used in stack reversals, undo operations, and linked list manipulations.

8. (a) Write a recursive algorithm for the Tower of Hanoi problem and explain it?
(b) Compare the trade-offs between recursion and iteration?
ans) Algorithm for Tower of Hanoi:

toh(n, source, auxiliary, destination):

```
if n == 1:
  move disk from source to destination
else:
  toh(n-1, source, destination, auxiliary)
  move disk from source to destination
toh(n-1, auxiliary, source, destination)
```

Explanation:
- the problem involves moving n disks from one peg to another using an auxiliary peg.
Step 1: Move n-1 disks from source to auxiliary.
Step 2: Move the remaining disk to the destination.
Step 3: Move the n-1 disks from auxiliary to destination.

For n=3, the total moves are $7 (2^n - 1)$.
time complexity is $O(2^n - 1)$.

Comparison of Recursion and Iteration:
- recursion uses system stack for each call, while iteration uses loop control variables.
- recursion is easier to write and understand but less efficient in memory and speed.
- iteration executes faster and uses less memory but may be harder to design for complex

problems.
- Some recursive problems can be easily converted to iterative versions, but not all.

9. Case Study - Design an efficient method using merge sort to sort an array. Explain its time complexity and benefits over bubble sort.
ans)
Concept: - Merge sort follows the divide and conquer approach. It divides the array into two halves, sorts each half recursively, and merges the sorted halves into one sorted array.

Algorithm:
MergeSort(A, l, r):
if l < r:
m = (l + r) // 2
MergeSort(A, l, m)
MergeSort(A, m + 1, r)
Merge(A, l, m, r)

Merge Procedure:
Merge(A, l, m, r):
i = l, j = m+1, k = l
while i <= m and j <= r:

if A[i] <= A[j]:
B[k++] = A[i++]
else:
B[k++] = A[j++]
copy remaining elements from A[i..m] or A[j..r] into B
copy B[l..r] back to A[l..r]

Example: For array [5, 2, 4, 6, 1, 3], merge sort repeatedly divides it and merges it into [1, 2, 3, 4, 5, 6].
Time Complexity: Best, average, and worst case = O(n log n).
Space Complexity: O(n) due to auxiliary array.

Advantages over Bubble Sort:
- Merge sort is faster for large datasets because bubble sort takes $O(n^2)$ time.
- it is stable and suitable for linked lists and external sorting.
- it guarantees consistent performance regardless of input distribution.
- Bubble sort is simple but inefficient for large inputs, while merge sort is scalable and widely