Set - 4

2. Row - major order - This method stores array row by row. The entire first row is stored in memory, followed by next row & so on

Column - major order - This method stores array column by column. The entire first column is stored in memory, followed by entire next column & so on.

Example For 2x3 array A = [1, 2, 3], [4, 5, 6]

Row major Layout [1, 2, 3, 4, 5, 6]

Column major layout [1, 4, 2, 5, 3, 6]

3. For 2-D array

B = Base address

S = Size of element

(i) Get to correct row = $(i \times N)$

(ii) Get to correct element = j

Address $A[i][j] = B + (i \times N + j) \times S$

For 3-D array

(i) Get to correct depth = $k \times D_1 \times D_2$

(ii) Get to correct column with depth = $j \times D_1$

(iii) Get to correct row = i

Address $A[i][j][k] = B + [(k \times D_1 \times D_2) + (j \times D_1) + i] \times S$

4.

Complexity	Insertion sort	Selection sort
Best case	$O(n)$	$O(n^2)$
Average case	$O(n^2)$	$O(n^2)$
worst case	$O(n^2)$	$O(n^2)$

5. Divide

(i) Divide unsorted array into two halves. Recursively calls itself until single sorted element is left in sub-array

(ii) Merge first compares two sub-arrays, then it copies the smaller element into temporary array first & then the larger number.

[8, 3, 5, 1]

[8, 3] & [5, 1]

[8] & [3] & [5] & [1]

compare 8 & 3
copy 3 then copy 8
[3, 8]

compare 5 & 1
copy 1 into temporary array then copy 5
[1, 5]

Merge [3, 8] & [1, 5]

Compare 3 & 1, copy 1 (Temp → [1])
compare 3 & 5, copy 3 (Temp → [1, 3])
compare 8 & 5, copy 5 (Temp → [1, 3, 5])
copy remaining 8 (Temp → [1, 3, 5, 8])

6. Removal of recursion is process of converting a recursive function into iterative on to —

(i) Avoid stack overflow

(ii) Improve performance

Iterative algorithm for factorial

```
int fact(int n){
    int result = 1;
    for(int i=n; i>0; i--){
        result = result * i; }
    return result; }
```

8.a) Base case - If 1 disk (n = 1), move directly from source to destination

Recursive case (for n disk(s)

(i) Move n-1 disks from source to temp (using desti as helper)

(ii) Move n^{th} disk from source to desti

(iii) Move n-1 disk from temp to desti (using source as helper)

```
TOH(n, source, desti, temp)
    If n == 1 then
Move disk 1 from source to desti

Else
// 1. Move n-1 from source to temp
    TOH(n-1, source, temp, desti)
// 2. Move largest from source to desti
    Move disk n from source to desti
```

// 3. Move n-1 from temp to desti
TOH(n-1, temp, desti, source)
END If

b) Advantages
(i) Code elegance - provide clean, simple & more readable solution
(ii) Problem decomposition - Break complex problem to smaller, identical sub-problems.

Disadvantages
(i) Memory usage - It can cause stack overflow error
(ii) Performance overhead - Function calls are more time taking.

9. Sparse matrix - Matrix which have mostly zeroes as its elements.

2-D array representation

2-D array of size (k+1) x 3
where k = no. of non-zero element
(i) First row [0] is header, storing original dimension & count [Total rows, total columns, Total non-zeroes]
(ii) Subsequent k rows each store a "triplet" for non-zero element [row, column, value]

Advantages –

(i) Space efficiency – For 1000 x 1000 matrix with only 100 non-zero element, the savings are 1,000,000 elements v/s 101x3 = 303 elements

Disadvantages –

(i) Slower access – will take time complexity O(k) instead of O(1)

(ii) Implementation complexity – Algorithms for matrices (add or multiply) becomes much complex as they must iterate through triplet instead using nested loops.