Date __/__/____

Set-4.

1. Define algorithm and explain impartance of efficiency in algorith design. 3M

⇒ An algorithm is step by step procedure or ~~of~~ set of well defined instructions to perform a specific task.

Example: Add to numbers.

⇒ Start

⇒ Read two numbers a, b.

⇒ calculate sum = a+b.

⇒ Display sum.

⇒ end.

Impartance of Efficiency in algo design.

⇒ When we design an algo, it's not enough that it just works - it should also be efficient. Efficiency means how fast an algorithm runs and how much space it takes in memory while executing.

i) Time Efficiency.
measures time taken by algo while executing.
Ex. ① Linear search O(n)
② Binary search O(log n).

ii) Space Efficiency.
measures how much memory (RAM) the algo uses.

⇒ A good algorithm is an algorithm which takes less time and less space in memory while executing.

SM

Date ___/___/_____

2. Explain row-major and column-major representation of arrays with suitable diagram. 3M

i) Row major :
In row major order, the element of a row are stored next to each other in memory.
That means -
The first row stored completely, then second, then third and so on ....

Ex. A[3][3] ⇒
$$\begin{bmatrix} 00 & 01 & 02 \\ 10 & 11 & 12 \\ 20 & 21 & 22 \end{bmatrix}$$

Stored in memory ⇒

A[0][0], A[0][1], A[0][2], A[1][0], A[1][1] ....

ii) Column major order.
⇒ Column major order the elems are stored column wise means first, first column elem stored, then second, then third .....

Ex A[3][3]
$$\begin{bmatrix} 00 & 01 & 02 \\ 10 & 11 & 12 \\ 20 & 21 & 22 \end{bmatrix}$$

Stored in memory ⇒
A[0][0], A[1][0], A[2][0], A[0][1], A[1][1], A[2][1] ....

SM

Date ___/___/_____

3. Derive the general index formula for multi-D arrays. 4M

i) for Row major order.

Add of A[I][J][K] =

Add A[I][J][K] = B + W x (N x L (I - x) + L x (J - y) + (K - z))

B = Base add
W = elem size
N = Total no of rows per layer.
L = Total no of cols per row.
x, y, z = lower bound of row, col, layer.

ii) for column major order.

Add A[I][J][K] = B + W * ((I - x) + N (J - y) + m (K - z))

N = Total no of layers
m = total number of rows.
x, y, z = lower bound of row, col, layer
if not specified assume 0.

SM

Date ___/___/_____

4. Compare insertion sort and selection sort with respect to their Time complexities. 4M.

⇒ Insertion sort works shifting elems it compares an elem with its previous elems and shifts them right by 1 position if they are greater than ~~current~~ key.

T.C:

(i) Best case O(n)
When the array is already sorted - only 1 comparision per elem.

(ii) Average case $O(n^2)$
When elements are in random order.

(iii) Worst case $O(n^2)$
When array sorted in reverse order. maximum number of comparision and shifts.

SC = O(1) (in place sorting).

ii) Selection sort:

⇒ selection sort repeatedly finds the smallest elem from the unsorted part and places it at the correct position in the sorted part.

TC:

(i) Best case $O(n^2)$
Even if array is already sorted selection sort still compares all elements.

(ii) Avg case $O(n^2)$
Finds minimum in each pass.

(iii) Worst case $O(n^2)$

→ Always performs same no of comparisions.

SC = O(1) (In place sorting)

Date ___/___/_____

5. Explain merge sort algorithm with an example. 5M

⇒ Algorithm!

1. Divide the array in two parts.
2. Recursively sorts each half.
3. Merge two sorted parts into single sorted array.

Pseudocode!

→ Merge Sort (arr, low, high)
→ if low < high then
→ mid = (low + high)/2
→ Merge sort (arr, low, mid)
→ Merge sort (arr, mid+1, high)
→ merge (arr, low, mid, high).

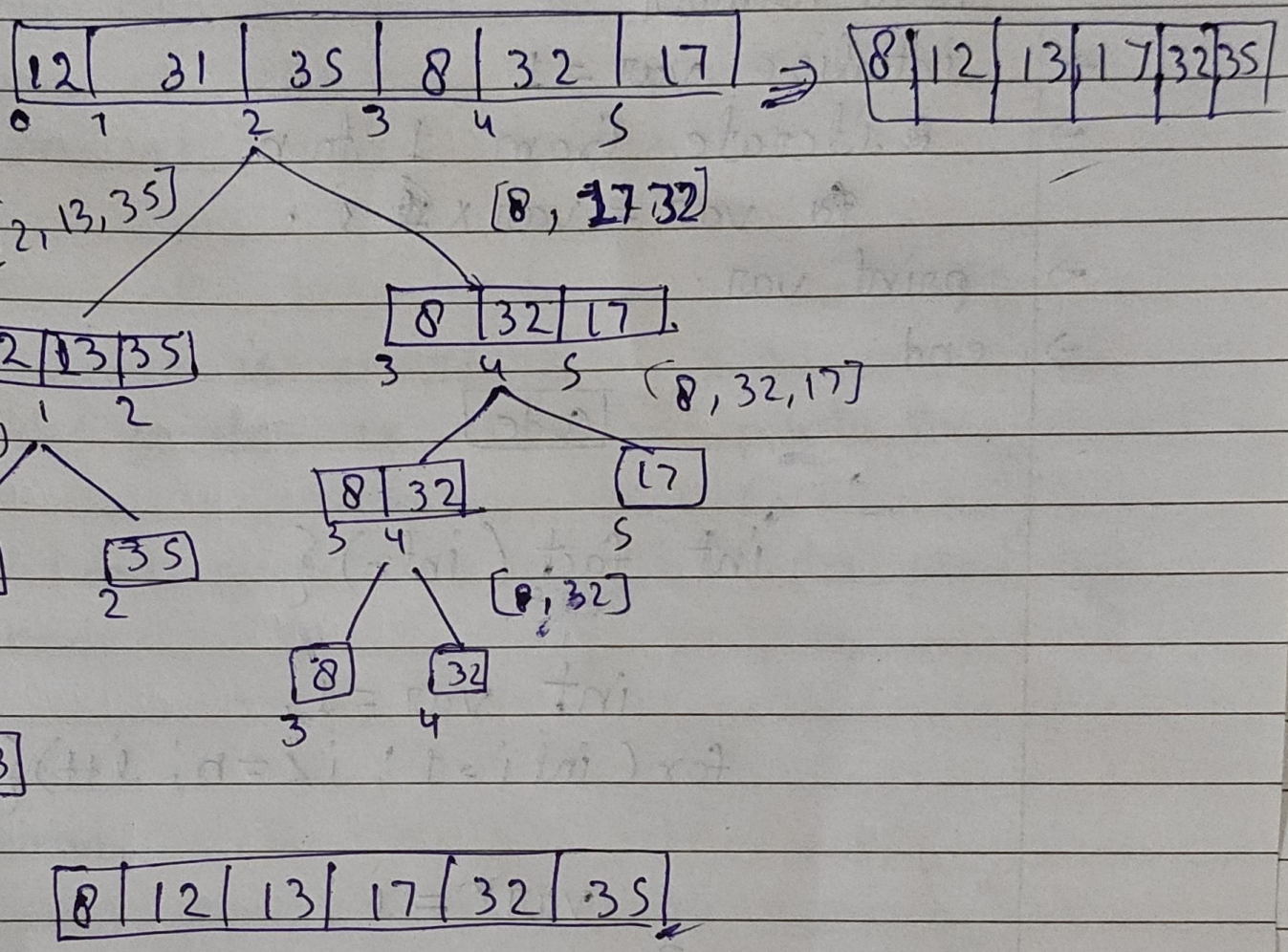
⇒ Merge (arr, low, mid, high)
⇒ Crete two arrays.
→ Left[] = arr [low → mid];
→ Right[] = arr [mid+1 → high]
→ i = 0, j = 0, k = low
→ while both subarrays have elm.
if Left[i] ≤ Right[j] then
arr[k] = Left[i]
i = i+1
else
arr[k] = Right[j]
j = j+1
K = K+1
⇒ Then copy remaing element ~~for both~~ or in as it is if elems left in left or right array..

SM

Date ___/___/_____

Example!

arr = | 12 | 31 | 35 | 8 | 32 | 17 |




| 8 | 12 | 13 | 17 | 32 | 35 |

6. Discuss removal of Recursion and write an iterative algorithm equivalent to the recursive factorial fun. 5M.

⇒ Removal of Rec!

⇒ Although recursion makes program shorter and easier to understand, but it has some drawbacks.

⇒ Every recursive call is stored in the function call stack, using more memory.

⇒ Slower execution.

⇒ Deep recursion can exceed the stack limit.

⇒ Harder to debug.

⇒ So, to improve efficiency, recursion can be converted into iteration.

SM

Date __/__/____

⇒ Iterative algo to find factorial ⇒

→ Start
→ Read number
→ initialize var = 1
→ iterate from 1 to n
    var = var * i.
→ print var.
→ end.

Code

```
int fact (int n) {

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

7. Explain insertion operation in singly linked list with example. 8 M

⇒ Insertion operation in linked list means adding new node in linked list. Each node contain two parts

1. Data → value of the node
2. Next → address of next node.

Date __/__/____

There are mainly three types of insertion in Linked List.

1. Insertion at begining
2. Insetion at end
3. Insetion at specific position.

⇒ Insertion at begining!

⇒ A new node is created.

→ Its next pointer is made to point the current head.

→ Then head pointer is updated to point this new node.

Algo1

→ Read Head

→ Declare var temp of Node pointer type.

→ Create a new node store its add in temp.

→ Now move Head in temp→next.

→ ~~Head~~ move temp in head.

→ End.

code

```
void insertBegining(Node* &head, int val){
    Node* temp = new Node(val);
    temp->next = head;
      head = temp;
}
```

SM

Date __/__/____

2. Insert at end

⇒ A new node created
⇒ Find last node
⇒ Link new node with it.

Algorithm:

⇒ Input Node Head and value.
⇒ Create a new pointer and assign tail = head;
⇒ Iterate tail and find last node.
⇒ Create a new node temp pass the value.
⇒ assign temp in tail->next.

Code

```
void insertAtEnd(Node* Head, int val) {
    Node* tail = Head;
    while (tail->next != nullptr) {
        tail = tail->next;
    }

    Node* temp = new Node(val);
    tail->next = temp;
}
```

3. Insertion at specific pos:

⇒ Traverse node upto given pos.
⇒ Create a new node.
⇒ Connect it with previous and next node.

SM

Date __/__/____

Algorithm:

⇒ Input Head, position, element.
⇒ Create temp node and temp = Head.
⇒ iterate temp for 1 to position -1 (means move temp to position -1).
⇒ Create new node temp2 and assign given value.
⇒ Now assign temp2 → next = temp → next.
⇒ and temp → next = temp2.
⇒ End.

```
void insert(Node *Head, int n, int elem){

    Node * temp = Head;
    for(i=1; i<n; i++){
        temp = temp -> next;
    }

    Node * temp2;
    temp2 = new Node(elem);
    temp2 -> next = temp -> next;
    temp -> next = temp2;
}
```

8(a) Write a recursive solution of Tower of Hanoi problem and explain the logic. 4M

⇒ Algorithm:
(i) Start
(ii) if n=1 then
    ⇒ move disk 1 from source to Destination
    Go to step (vi) = 6

Date __/__/____

(iii) call the function( n-1, Source, Destination, Helper)
(iv) move disk n from Source → Dest.
(v) call the function(n-1, Helper, Source, Destination)
(vi) end.

**Code**

```
void towerH (int n, char source, char helper,
                    char dest) {

    if (n==1) {

        cout << "Move disk 1 from" <<source<< "to"
                << destination << "\n";

        return
            }

    towerH (n-1, source, dest, helper);

cout << "Move disk" <<n << "from" << source<< "to"
                << dest << "\n";

    towerH(n-1, helper, source, dest);
}
```

Date __/__/____

⇒ Logic Explanation!

⇒ to move n disks, first move the top n-1 disks to helper rod(B).

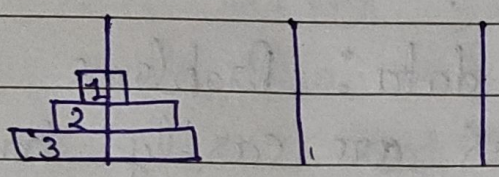
⇒ Then move the largest disk (n-th) to dest (C).

⇒ Finally move the n-1 disks from helper rod(B) to dest.

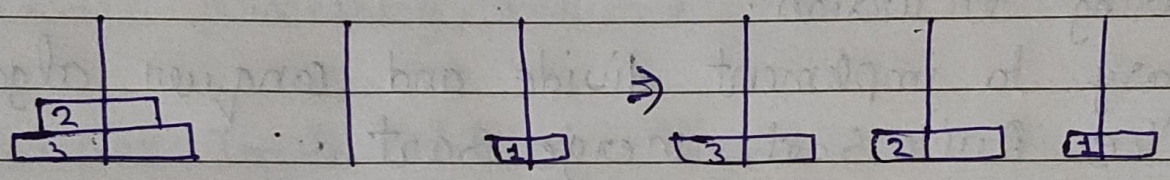
⇒ if n == 1 then directly move src to dest.

Ex n = 3

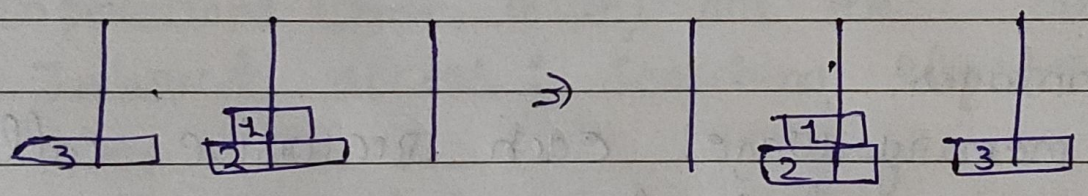
①



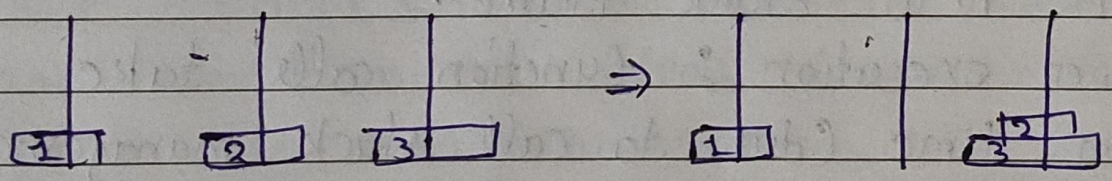

⑪



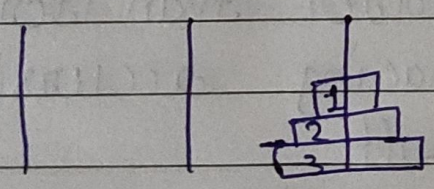

⇒




⇒




⇒




Number of moves ⇒ $2^n - 1$ ⇒ $2^3 - 1 = 7$ moves

Date __/__/____

8 (b) Discuss the advantages and disadvantages of recursion in problem salving. 4 M

Advantages:

⇒ Simpler and cleaner code: Complex problems like Tower of Hanoi, Fibbonacci etc solved in fewer lines of code.

⇒ Reduces the need for loops: Recursion replaces iterative looping structure makes logic easier to understand.

⇒ Natural for hierarchical data: Problems of tree, graphs, directories are easily handled using recursion.

⇒ Easy to implement divide and conquer algo's. like Quik Sort, merge sort.

Disadvantages:

⇒ High memory usage: each recursive call uses new stack frame.

⇒ Slower execution: function calls take extra time (due to call stack management) so recursion can be slower than iteration.

⇒ Difficult to Debug: Tracing recursive calls might be difficult.

⇒ Inefficient for simple tasks like reverse array factorial.

Date __/__/____

9. With example, explain sparse matrix representation using 2D array and discuss its advantages and limitations. 10 M

Spars matrix → Set 1 Q6

⇒ Its advantages and Limitations:

① Memory efficient: Saves space by storing only nonzero elements.
② Faster computation: Arithmetic operations (add, trans) becomes faster since we have fewe elems.
③ Easy to store and transfer
④ Useful in scientific and data applications. like image processing, network graph, ML.

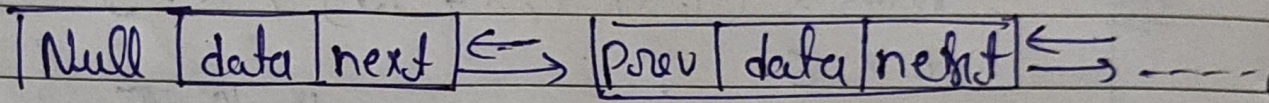
① Complex Implementation: Algorithms for matrix operations (Add, mult) are more complicated.
② Indirect access: Searching element more complicated.
③ Not efficient for dense matrices: If there are many non-zero elems sparse matrix may take more memory and time.
④ Extra Overhead: Each nonzero value needs to store its row and col info.

Date __/__/____

9 (OR) Discuss the use of doubly linked lists in complex data management scenarios, with example of traversal and deletion operations. 10 M.

⇒ A Doubly Linked List is a data structure where each node contains three fields:

1. prev → pointer to the previous node.
2. data → the actual data stored in the node.
3. next → pointer to the next node.




Use in Complex Data Management scenarios:

⇒ Doubly liked lists are useful in applications where frequent insertions, deletions, and bidirectional traversal required.

1. Web Browser
⇒ Each webpage is a node.
⇒ The prev pointer helps to go backward and next pointer helps to go forward between pages.
2. Music or Video Playlists
⇒ Songs or videos are sorted as nodes.
⇒ Users can move to the previous or next item easily.
3. Undo - Redo functionalities in text Editors.
⇒ Each action is a node
⇒ Moving backward and forward is easy using prev and next pointer.

SM

Date __/__/____

4. Memory Management System.
Operating Systems Use DLLs to maintain free and allocated memory blocks efficiently.

5. Navigation System or Games
⇒ Allows moving back and forth b/w different states or levels.

Traversal and deletion operations
Set 1 Q 9 OR.