1) Matrix multiplication:

Aim:

To perform multiplication of two matrices and display the result.

Algorithm:

Step 1: Start the Program

Step 2: Input rows and columns for Matrix-A

Step 3: Input rows and columns for Matrix-B

Step 4: check if columns of A = rows of B

Step 5: Input elements for Matrix A and B

Step 6: Initialize result matrix with 0

Step 7: Multiply using nested loops

Step 8: Display result matrix

Step 9: End the Program

Sample input:

Matrix A: 2×3, Matrix B: 3×2

A = [1 2 3], [4 5 6]

B = [7 8], [9 10], [11, 12]

Sample output:

Result = [58 64], [139 154]

Result:

Matrix multiplication performed successfully

2) Odd (or) Even from a set of number

Aim:

To find and seperate odd and even numbers from a given list algorithm:

Step 1: Start

Step 2: Input number of elements.

Step 3: Input elements into an array.

Step 4: For each number, check if it is divisible by 2.

Step 5: If yes, it's even; else odd

Step 6: Display all odd and even numbers

Step 7: End.

Sample Input

List: 10, 21, 32, 43, 54.

Sample Output

even: 10, 32, 54

odd: 21, 43.

3. Factorial (without recursion)

Aim:

To calculate factorial of a number using loop

Algorithm:

Step-1 : Start

Step-2 : Input a number.

Step 3 : Initialize result = 1

Step 4 : Multiply result by all numbers from 1 to 4

Step 5 : Display result.

Step-6 : End.

Sample input :

5

Sample output :

120.

Result :

Factorial calculated succesfully using loop

9) Fibonacci series (without recursion):

Aim:

To print Fibonacci series upto 'n' terms using loop.

Algorithm:

Step-1: Start

Step-2: Input number of terms.

Step 3: Initialize first = 0, second = 1

Step-4: loop to print next = first + second

Step-5: Update first and second values.

Step-6: Repeat and display series.

Step-7: End.

Sample input

5.

Sample output

0 1 1 2 3.

Result:

Fibonacci series displayed successfully.

5) Factorial (using recursion)

Aim:

To find factorial using a recursive function

Algorithm:

Step-1: start

Step-2: Input a number n

Step-3: Define recursive function

Step-4: if n == 0 (or) 1, return 1

Step-5: Else return n, factorial(n-1)

Step-6: Display result.

Step-7: End.

Sample Input:

5.

Sample output:

factorial = 120.

Result:

Factorial calculated successfully using recursion

6) Fibonacci series (using recursion)

Aim:

To generate fibonacci series using recursion.

Algorithm

Step-1: start

Step-2: Input number of terms.

Step-3: Define recursive function

Step-4: If n==0 return 0; if n==1, return 1

Step-5: Else return fib(n-1)+fib(n-2)

Step-6: call function and display results.

Step-7: End.

Sample Input:

5

Sample output:

0 1 1 2 3.

Result:

Fibonacci series generated successfully using recursion.

7. Array operations (Insert, Delete, Display)

Aim:

To Perform insert, delete and display operations on an array

Algorithm:

Step-1: start

Step-2: Input size and elements of array.

Step-3: For insert: input Position and value, Shift and insert.

Step-4: for delete: input Position and value Shift elements to remove.

Step-5: for display: Print all elements

Step-6: End

Sample INPut:

Array: [10, 20, 30], Insert 25 at Pos 2, delete from Pos 1.

Sample output:

After insert: [10, 20, 25, 30]

After delete: [20, 25, 30]

Result:

Array operations Performed Successfully.

8) Linear Search:

Aim:

To search an element in a array using linear search

Algorithm:

Step 1: start.

Step 2: Input array and elements to search.

Step-3: Traverse array from start to end

Step-4: If match found, Print Position

Step 5: Else Print, not found.

Step 6: End.

Sample INPUT:

Array: [10,20,30,40], search: 30.

Sample output:

Element found at Point: 3

Result:

Element searched successfully using linear search.

9) Binary search:

Aim:
To search an element using binary search in a sorted array

Algorithm:
Step-1: start
Step-2: Input sorted array and element to search.
Step-3: set low = 0, high = n-1
Step-4: while low high - calculate mid.
Step 5: compare mid element, adjust low / high
Step 6: If found, Print position else not found.
Step-7: End

Sample Input:
Array = [10, 20, 30, 40, 50], search = 40.

Sample output:
element found at Position: 4

Result:
Element searched successfully using binary search.

10. Linked list operations:

Aim:

To implement basic operations on a linked list like insert, delete, and display.

Algorithm:

Step-1: Start

Step-2: create a node structure with data and next.

Step 3: Insert node; set data and update links.

Step 4: Delete node: locate and unlink node.

Step 5: Display list: traverse and Print.

Step-6: End.

Sample Input:

Insert 10,20,30; Delete 20.

Sample output:

List after insert = 10,20,30

List after delete = 10,30

Result:

Linked list operations performed Successfully.

11. Stack operations : Push, POP, PEEK

Aim:

Implement a stack with basic operations : PUSH, POP and PEEK

Algorithm:

Step 1: Start

Step 2: Initialize an empty stack

Step 3: Push add an element to the top of stack

Step 4: POP remove top element from the Stack

Step 5: PEEK : view the top element without removing it.

Step 6: Repeat operations based on user input.

Step 7: End

input :

PUSH: 10

PUSH: 20

PEEK

POP

output :

TOP element : 20

Popped : 20

Result :-

Stack operations based on user successfully implemented.

12. Application of stack

Aim:

Convert infix expressions to postive using a Stack

Algorithm:

Step 1: Start

Step 2: Initialize an empty Stack for operators

Step 3: Read the infix expression from left to right.

Step 4: if the character is an operand add it to the Output.

Step 5: After reading the expression, POP all remaining operators to the output.

Sample input:

A+B+C.

Sample output:

Postfix: AB C * +

Result:

Notation conversion was Successfully implemented.

15. Queue operations.

Aim:

implement a queue operation ENQUEUE, DEQUEUE, and Display

Algorithm:

Step 1: initialize front and rear Pointers to -1

Step 2: Check for overflow, if empty, set front and rear to 0.

Step 3: check for underflow, return the element at front and increment front.

Step 4: a traverse and print element to from front to rear.

Sample input:

Enqueue :10.

Enqueue :20

Dequeue :

Display

Sample output:

Dequeued:10.

Queue :-20.

Result:

Queue operations were sucessfully implemented.

6 Tree Traversals

Aim:
implement tree traversals: inorder, preorder, and Postorder.

Algorithm:

Step 1: Start

Step 2: Create a binary tree

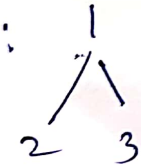
Step 3: inorder Traversal: a. Traverse left Subtree, b. visit root, c. Traverse right Subtree.

Step 4: Preorder Traversal: a. visit root. b. Traverse left Subtree. c. Traverse right Subtree.

Step 5: Postorder Traversal: a. Traverse left Subtree. b. Traverse right Subtree. c. visit root

Sample input:

Tree:



Sample output:

inorder: 2 1 3
Preorder: 1 2 3
Postorder: 2 3 1

Result:

Tree traversal was successfully implemented.

15. Hashing using Linear Probing

Aim:

implement hashing with linear probing.

Algorithm:

1. Step 1: create a hash table of fixed size
2. Step 2: for each key: a. compute hash index = key % table size. b. If index is occupied probe to next index linearly until an empty spot is found. c. insert key

Sample input:

Keys: 10, 20, 50

Table size: 10

Sample output:

Hash table:

index 0: 10

index 1: 20

index 2: 30

Result:

Hashing with Linear Probing was Successfully implemented.

16. Insertion sort

Aim:

Sort an array using insertion sort

Algorithm:

Step 1: Liberate from index 1 to end.

Step 2: for each index : a. store current value. b. compare with elements on the left. c. Shift larger elements to the Right d. Insert current value in correct position

Sample input:

Array = [5, 2, 4, 6, 1, 3]

Sample output:

Sorted Array: [1, 2, 3, 4, 5, 6]

Result:

Insertion sort was successfully implemented.

17. Merge sort.

Aim:

Sort a array using merge sort.

Algorithm:

Step 1: Divide the array into halves recursively.

Step 2: Merge two sorted halves: a. compare elements from both halves. b. copy smaller element into new array.

Step 3: Continue until all element are merged.

Sample Input:

array = [38, 27, 43, 3, 9, 82, 10]

sample output:

Sorted array: [3, 9, 10, 27, 38, 43, 82]

Result:

Merge sort was successfully implemented.

18. Quick sort.

Aim:

Sort an array using quick sort

Algorithm:

Step 1: Select a Pivot element.

Step 2: partition array: a. Place pivot in correct position: b. Elements less than Pivot to left: c. Elements greater to right.

Sample Input:

Array: [10, 7, 8, 9, 1, 5]

Sample output:

Sorted Array: [1, 5, 7, 8, 9, 10]

Result:

Quick sort was successfully implemented.

19. Heap Sort

Aim:

Sort an array using heap sort.

Algorithm:

Step 1: Build a max heap from the array

Step 2: Swap first element with last

Step 3: Reduce heap size and heapify root

Step 4: Repeat until sorted

Sample input:

Array: [12, 11, 13, 5, 6, 7]

Sample output:

Sorted Array: [5, 6, 7, 11, 12, 13]

Result:

Heap Sort was successfully implemented

20. AVL tree: Insert, Delete, Search.

Aim:

implement AVL tree operations: insert, Delete and search.

Algorithm:

Step 1: Insert node like in BST.

Step 2: Check balance factor.

Step 3: Perform Rotation if unbalanced.

Step 4: Dellete node like in BST.

Step 5: Rebalance using rotation.

Step 6: Search node like in BST.

Sample input:

Insert: 10, 20, 30

Delete: 20.

Search: 30.

Sample output:

Search found: 30.

Result:

AVL tree operations were Sucessfully Implemented.

21. GRAPH TRAVERSAL

Aim:

Traverse graph using Breadth First search

Algorithm

Step 1: use a queue and visited array

Step:2 : inqueue start node.

Step3: while queue not empty : a Dequeue node b. visit and enqueue allunvisited neighbours.

Sample input:

Graph: 0->1, 2; 1->2; 2->0, 3; 3->3.

Start: 2.

Sample output:

BFS: 2031.

Result:

BFS traversal was successfully implemented.

22. Graph Traversal.

Aim:

Traverse graph using Depth First Search.

Algorithm:

Step 1: use a stack or recursion and visited array.

step 2: Start from node and visit recursively; a mark current as visited R: recur for all unvisited neighbours.

Sample input:

Graph: 0 -> 1, 2; 1 -> 2; 2 -> 0, 3; 3 -> 3

start: 2.

Sample output:

DFS: 2013.

Result:

DFS traversal was successfully implemented.

23. DIJKSTRA'S ALGORITHM

Aim:

Find shortest Path using Dijkstra's algorithm.

Algorithm:

Step 1: Set all distances to infinity; source=0.

Step 2: use a min-Priority queue.

Step 3: while queue not empty: a: Extract min node: b: for each neighbour, if shorter path found, update distance.

Sample input:

Graph: 0 → (1,4), (2,1) : 1 → (3,1); 2 → (1,2) (3,5)

Source: 0.

Sample output:

Distance: 0 1 3 4

Result: Dijkstra's algorithm way successfully implemented.

24. PRIMS Algorithm :

Aim:

- Find MST using Prim's Algorithm.

Algorithm:

Step 1: initialize all keys as intinty: MstSet[] = False

Step 2: Start from 0, key[0]=0,

Step 3: Repeat (V-1) times: a. Pick min key not MST. b. include in MST. c. Update key of adjacent nodes.

Sample Input:

Graph = Weighted undirected with 5 vertices.

Sample output:

MST Edges: 0-1, 1-1, 1-4, 0-3.

Result:

Prints algorithm was successfully implemented.

25. KRuskal's algorithm.

Aim:

Find MST using Kruskal's algorithm

Algorithm:

Step 1: Sort all edges by weight

Step 2: initialize disjoint sets

Step 3: For each edges: a if its does'nt form a cycle, include it. b. Union the sets.

Sample input:

Edges: (0-1, 10), (0-2, 6), (0-3, 5), (1-3, 5) (2-3, 4)

Sample output:

MST Edges: 2-3, 0-3 01

Result:

Kruskal's algorithm was successfully implemented now about i get all those inside format for my convenient.