EXPERIMENT : No.

Page No. Date

Data Structures (unit 1)

1) Array operations

• Traversing An Array

→ Step 1 :- [initialization] set I = lower_bound
Step 2 :- Repeat Step 3 to 4
while I <= upper_bound
Step 3 :- Apply process to A[I]
Step 4 :- Set I = I + 1
[END OF LOOP]
Step 5 :- EXIT

• inserting an element in Array

→ Step 1 :- Set upper_bound = upper_bound + 1
Step 2 :- Set A [upper_bound] = val
Step 3 :- EXIT

(Algo. to append a new element to an existing array)

→ Step 1 :- [initialization] set I = N
Step 2 : Repeat 3 and 4 while I >= POS
Step 3 : set A [I + 1] = A [I]
Step 4 : set I = I + 1 [END OF LOOP]
Step 5 : set N = N + 1
Step 6 : set A[POS] = VAL
Step 7 : EXIT.

(add new element to middle of an array)

- Delete element in a Array:-

→ Step 1: set upper_bound = upper_bound - 1
Step 2: EXIT

(Algo. to delete last element of an Array)

→ Step 1: [initialization] set I = POS
Step 2: Repeat step 3 and 4 while I<=N-1
Step 3: Set A[I] = A[I+1]
Step 4: Set I = I+1 [END OF LOOP]
Step 5: Set N = N-1
Step 6: EXIT

(Algo to delete an element from the middle of an Array)

2) Linear search:-

→ Step 1: [initialize] set POS = -1
Step 2: [initialize] set I = 1
Step 3: Repeat Step 4 while I <= N
Step 4: IF A[I] = VAL
            SET POS = I
            PRINT POS
            Go TO step 6
        [END OF IF]
        set I = I + 1
    [END OF LOOP]
Step 5:- IF POS = -1
        PRINT "VALUE IS NOT PRESENT IN AN ARRAY"
        [END OF IF]
Step 6: EXIT.

EXPERIMENT : No. Page No. Date

3) Binary Search:-

→ Step 1: [initialize] Set BEG = lower_bound
END = upper_bound , POS = -1

Step 2: Repeat Step 3 and 4 while BEG <= END

Step 3: Set MID = (BEG + END) / 2

Step 4: IF A [MID] = VAL
SET POS = MID
PRINT POS
GO TO Step 6
ELSE IF A[MID] > VAL
SET END = MID -1
ELSE
SET BEG = MID + 1
[END OF IF]
[END OF LOOP]

Step 5 : IF POS = -1
PRINT "VALUE IS NOT PRESENT IN
IN THE ARRAY" [END OF IF]

Step 6: EXIT

4) Sorting:-

→ Step 1: Repeat Step 2 for 1 = N -1

Step 2: Repeat For J = to N-1

Step 3: IF A [J] > A[J + 1]
SWAP A[J] and A[J +1]
[END OF INNER LOOP]
[END OF OUTER LOOP]

Step 4: EXIT

5) Insertion Sort:-

→ Step 1: Repeat Step 2 to 5 for k = 1 to N-1

Step 2: SET Temp = arr[K]

Step 3: Set J = K-1

Step 4: Repeat while Temp <= arr[J]

set arr[J+1] = arr[J]

set J = J-1

[END OF INNER LOOP]

Step 5: set ARR[J+1] = Temp

Step 6: EXIT

6) Selection Sort:-

- Smallest (ARR, K, N, POS):-

→ Step 1:- [initialize] set small = Arr[K]

Step 2:- [initialize] Set pos = K

Step 3: Repeat for J = K+1 to N-1

if small > arr[J]

set small = Arr[J]

set pos = J

[END OF IF]

[END OF LOOP]

Step 4: RETURN POS

- Selection (ARR, N):-

→ Step 1:- Repeat Step 2 and 3 for K = 1 to N-1

Step 2:- Call Smallest (Arr, K, N, POS)

Step 3:- Swap A[K] with ARR[POS]

[END OF LOOP]

Step 4:- EXIT.

UNIT NO:- 2

1) Push Operation:-

→ Step 1: if TOP = MAX - 1
PRINT "OVERFLOW"
GOTO Step 4
[END OF IF]
Step 2: Set TOP = TOP + 1
Step 3: Set Stack [TOP] = value
Step 4: END

2) Pop operation:-

→ Step 1: if top = NULL
PRINT "UNDERFLOW"
Goto Step 4 [END OF IF]
Step 2: Set Val = Stack [TOP]
Step 3: Set TOP = TOP - 1
Step 4: END.

3) Peek Operation:-

→ ~~Step 4~~ Step 1: if top = NULL
PRINT "Stack is empty"
Goto step 3.
Step 2: Return Stack [TOP]
Step 3: END

UNIT NO :- 03 (Queue)

1) insert array elements in Queue :-

→ Step 1: if rear = max -1
write "overflow"
GOto step 4
[END OF IF]

Step 2: if front = -1 and rear = -1
set front = rear = 0
else
set rear = rear +1
[end of if]

Step 3: set QUEUE [REAR] = NUM

Step 4: EXIT

2) Deletion in Queue :-

→ Step 1:- if front = -1 OR FRONT > REAR
WRITE UNDERFLOW
ELSE
SET VAL = QUEUE [FRONT]
SET FRONT = FRONT +1
[END OF IF]

Step 2: EXIT.

3) Circular Insert Algorithm :-

→ Step 1: if (front == 0 AND Rear == MAX -) OR
(front == Rear +1)
PRINT OVERFLOW
Goto step 5 [END OF IF]

Step 2: if front == -1
set front = Rear = 0
else IF, Rear == MAX -1
set Rear = 0
else
set Rear = Rear + 1 [end of IF]

EXPERIMENT : No.

Page No.
Date

Step 3 : set QUEUE [Rear] = NUM

Step 4 : Print "element inserted"

step 5 : EXIT.

4) Circular Delete Algorithm :-

→ Step 1 :- if front = = -1
write UNDERFLOW
GOTO step 4 [END OF IF]

Step 2 :- set val = QUEUE [FRONT]

Step 3 :- if front = = Rear
set front = Rear = -1
else IF front = = MAX - 1
Set FRONT = 0
else
set Front = Front + 1
[END OF IF]

Step 4 : EXIT.

5) Dequeue : insert :-

• Rear

→ Step 1 :- if (front = = 0 and Rear = = Max - 1)
OR (FRONT = = Rear + 1)
write OVERFLOW
EXIT

Step 2 :- if FRONT = = -1
set FRONT = REAR = 0
else IF REAR = = MAX - 1
set REAR = 0
else

Teacher's Sign. :

Set REAR = REAR + 1

Step 3: Set Dequeue [Rear] = Value

Step 4: EXIT.

- **Front**

→ Step 1:- if (Front = = 0 and Rear = = MAX - 1) OR
(Front = = Rear + 1)
write OVERFLOW
EXIT

Step 2: if FRONT = = -1
Set FRONT = REAR = 0
else IF FRONT = = 0
set FRONT = MAX - 1
else
set FRONT = FRONT - 1

Step 3: set Dequeue [FRONT] = value

Step 4: EXIT.

6) Dequeue: Deletion.

- **Rear**:-

→ Step 1:- if Front = = -1
write underflow
EXIT

Step 2: Set value = Dequeue [Rear]

Step 3: if FRONT = = REAR
Set FRONT = = REAR
else IF REAR = = 0
Set REAR = MAX - 1
else
set REAR = Rear - 1

Step 4: EXIT.

EXPERIMENT : No. Page No. Date

- **Front :-**

→ Step 1 :- if front == -1
    write UNDERFLOW
    EXIT

Step 2 : set value = Deque[FRONT]

step 3 :- if front == Rear
    set front = Rear = -1
  else if front == MAX-1
    set front = 0
  else
    set front = front + 1

Step 4 :- EXIT.

1) Traversing a Linked List :-

→ Step 1 :- [initialize] set PTR = START

Step 2: Repeat Step 3 and 4
while PTR != NULL

Step 3 :- Apply Process - to PTR -> DATA

Step 4 :- Set PTR = PTR -> NEXT
[END OF LOOP]

Step 5 :- EXIT.

2) To print the number of Nodes in a Linked List :-

→ Step 1 :- [initialize] set count = 0

Step 2 :- [initialize] set PTR = START

Step 3 :- Repeat step 4 and 5
while PTR != NULL

Step 4 :- Set count = count + 1

Step 5 :- Set PTR = PTR -> NEXT
[END OF LOOP]

Step 6 :- WRITE COUNT

Step 7 : EXIT

3) Searching for a value in the Linked List :-

→ step 1 :- [initialize] set PTR = START

Step 2 :- Repeat Step 3 while PTR != NULL

Step 3 :- If val = PTR -> DATA
set POS = PTR
GOTO Step 5
ELSE
SET = PTR -> NEXT
[END OF IF] [END OF LOOP]

Step 4 :- Set POS = NULL

Step 5 :- EXIT.

EXPERIMENT : No.

Page No.
Date

4) inserting a node at beginning of the linked list :-

→ Step 1 :- If AVAIL = NULL
write OVERFLOW
GOTO step 7 [END OF IF]
Step 2 :- Set NEW-NODE = AVAIL
Step 3 :- Set AVAIL = AVAIL -> NEXT
Step 4 :- Set NEW-NODE -> DATA = VAL
Step 5 :- Set NEW-NODE -> NXT = START
Step 6 :- Set START = NEW-NODE
Step 7 : EXIT

5) inserting a node at end of the Linked List :-

→ Step 1 :- if AVAIL = NULL
write OVERFLOW
GO TO step 10 [END OF IF]
Step 2 :- Set NEW-NODE = AVAIL
Step 3 :- Set AVAIL = AVAIL -> NEXT
Step 4 :- set NEW-NODE -> DATA = VAL
Step 5 : Set NEW-NODE -> NEXT = NULL
Step 6 : Set PTR = START
Step 7 : Repeat step 8 while PTR -> NEXT != NULL
Step 8 : Set PTR = PTR -> NEXT [END OF LOOP]
Step 9 : Set PTR -> NEXT = NEW-NODE
Step 10 :- EXIT.

6) insert a new node after a node that has value NUM:-

→ Step 1:- if AVAIL = null
write OVERFLOW
GOTO Step 12 [END OF IF]

Step 2: set New_node = avail

Step 3: set avail = avail -> next.

Step 4:- set new_node -> data = val.

Step 5:- set PTR = START

Step 6:- set PREPTR = PTR

Step 7:- Repeat step 8 and 8
while PREPTR -> DATA != NUM.

Step 8:- set PREPTR = PTR

Step 9:- Set PTR = PTR -> NEXT

Step 10:- PREPTR -> NEXT = new_node

Step 11:- set new_node -> next = PTR

Step 12: EXIT.

7) inserting a Node before a given Node in Linked List:-

→ Step 1:- if avail = null
write OVERFLOW
Go to step 12 [END OF IF]

Step 2:- set new_node = avail

Step 3:- set avail = avail -> next

Step 4:- set new_node -> data = val

Step 5:- set PTR = START

Step 6:- set PREPTR = PTR

Step 7:- Repeat step 8 and 9
while PTR -> data != NUM

Step 8:- set PREPTR = PTR

Step 9:- set PTR = PTR -> NEXT

Step 10:- PREPTR -> NEXT = NEW_NODE

Step 11:- set NEW_NODE -> NEXT = PTR

Step 12:- EXIT

EXPERIMENT : No.

Page No.
Date

8) Deleting the 1st node of the linked list:-

→ Step 1:- if START = NULL
Write UNDERFLOW
Go to Step 5
[END OF IF]
Step 2:- Set PTR = START
Step 3:- Set START = START -> NEXT
Step 4:- FREE PTR
Step 5:- EXIT

9) deleting the last node from the Linked List:-

→ Step 1:- if START = NULL
Write UNDERFLOW
Go to Step 8 [END OF IF]
Step 2:- Set PTR = START
Step 3:- Repeat step 4 and 5
while PTR -> next != NULL
Step 4:- set PREPTR = PTR
Step 5:- Set PTR = PTR -> NEXT
[END OF LOOP]
Step 6:- set PREPTR -> NEXT = NULL
Step 7:- FREE PTR
Step 8: EXIT.

10) deleting the Node after a given node in Linked List:-

→ Step 1:- if Start = null
write UNDERFLOW
Goto step 10 [END OF IF]

Step 2:- set PTR = START

Step 3:- set PREPTR = PTR

Step 4:- Repeat steps 5 and 6
while PREPTR -> DATA != NUM

Step 5:- set PREPTR = PTR

Step 6:- Set PTR = PTR -> NEXT
[END OF LOOP]

Step 7:- set TEMP = PTR

Step 8:- Set PREPTR -> NEXT = PTR -> NEXT

Step 9:- FREE TEMP

Step 10:- EXIT.

11) Sort Linked List in Ascending order:-

→ Step 1:- if start = null or start -> ~~next~~ next = NULL
write "List is too short to sort"
Goto step 9 [END OF IF]

Step 2:- set PTR1 = START

Step 3:- Repeat Step 4 ~~and~~ to 7 while PTR1 ≠ NULL

Step 4:- Set PTR2 = PTR1 -> NXT

Step 5:- Repeat Step 6 and 7 while PTR2 ≠ NULL

Step 6:- if PTR1 -> DATA > PTR2 -> DATA
set TEMP = PTR1 -> DATA
set PTR1 -> Data = PTR2 -> DATA
Set PTR2 -> DATA = TEMP
[END OF IF]

Step 7:- Set PTR2 = PTR2 -> NEXT
[END OF INNER LOOP]

Step 8:- Set PTR1 = PTR1 -> NEXT
[END OF OUTER LOOP]

Step 9:- EXIT.

EXPERIMENT : No. Page No. Date

12) Merging the Node :-

→ Step 1 :- if list1 = NULL
set Merged_list = list2
Go to step 9 [END OF IF]

Step 2 :- if list2 = null
set merged_list = list1
Go to step 9 [END OF IF]

Step 3 :- initialize pointers
set PTR1 = list1
set PTR2 = list2
set merged_list = null
set Last = NULL

Step 4 :- Repeat Step 5-7 while PTR1 ≠ NULL &
PTR2 ≠ NULL

Step 5 :- if PTR1 -> Data <= PTR2 -> DATA
set Temp = PTR1
set PTR1 = PTR1 -> NEXT
ELSE
set TEMP = PTR2
Set PTR2 = PTR2 -> Next [END OF IF]

Step 6 :- if merged_list = NULL
set merged_list = temp
set last = temp
else
set last -> Next = temp
set last = temp [END OF IF]

Step 7 :- [END OF LOOP]

Step 8 :- If PTR1 ≠ NULL
set Last -> next = PTR1

Teacher's Sign. :

else if PTR2 ≠ NULL
set Last -> next = PTR2
[END OF IF]
Step 9:- EXIT

13) inserting a node at the ~~beginning~~ end of the Doubly Linked List :-

→ Step 1:- if avail = null
write overflow
Go to step 9 [END OF IF]
Step 2:- set new_node = avail
Step 3:- set avail = avail -> next
Step 4:- set new_node -> data = val
Step 5:- set new_node -> prev = null
Step 6:- set PTR = START
Step 7:- Repeat step 8 while PTR -> NEXT != NULL
Step 8:- set PTR = PTR -> NEXT [END OF LOOP]
Step 9:- set PTR -> next = new_node
Step 10:- set New_node -> prev = PTR
Step 11:- EXIT

14) inserting a node at the beginning of the Doubly Linked List :-

→ Step 1:- if avail = null
write OVERFLOW
Go to step 9 [END OF IF]
Step 2:- Set new_node = avail
Step 3:- set avail = avail -> next
Step 4:- set new_node -> data = val
Step 5:- set new_node -> prev = null
Step 6:- set new_node -> next = start
Step 7:- set start -> prev = new_node
Step 8:- set start = new_node
Step 9:- EXIT

15) Algorithm to insert a new node after a given node:-

→ Step 1 :- if avail = null
write OVERFLOW
Go to step 12 [END OF IF]

Step 2:- set new-node = avail

Step 3:- set avail = avail -> next

Step 4:- set new_node -> data = val

Step 5:- set PTR = START

Step 6:- Repeat step 7 while PTR ->Data != NUM

Step 7:- set PTR = PTR -> next [END OF LOOP]

Step 8:- set new_node -> next = PTR -> next

Step 9:- set new-node -> prev = ptr

Step 10:- set PTR -> next = new-node

Step 11:- set PTR -> next -> prev = new-node

Step 12: EXIT

16) Algorithm to insert a new node before a given node:-

→ Step 1 :- If avail = null
write OVERFLOW
Go to step 12
[END OF IF]

Step 2: set new_node = avail

Step 3: set avail = avail ->next

Step 4:- set new_node -> data = val

Step 5:- set PTR = START

Step 6:- Repeat step 7 while PTR → data != NUM.

Step 7 :- set PTR = PTR -> Next

Step 8 :- set new_node -> next = PTR

Step 9 :- set new-node -> prev = PTR -> prev

Step 10 :- set PTR -> prev = new-node

Step 11 :- set PTR -> prev -> next = new-node

Step 12 :- EXIT.

17) deleting the 1st node from a doubly linked list :-

→ Step 1 :- If start = null
write OVERFLOW
Go to step 6 [END OF IF]

Step 2 :- set PTR = START

Step 3 :- set START = START -> next

Step 4 :- set START -> PREV = NULL

Step 5 :- FREE PTR

Step 6 :- EXIT

18) deleting the last node from a doubly linked list :-

→ Step 1 :- If start = null
write OVERFLOW
Go to Step 7 [END OF IF]

Step 2 :- Set PTR = START

Step 3 :- Repeat Step 4 while PTR -> next != null

Step 4 :- set PTR = PTR -> next
[END OF IF]

Step 5 :- set PTR -> PREV -> next = null

Step 6 :- Free PTR

Step 7 : exit

EXPERIMENT : No. Page No. Date

19) deliting the node before a given node in doubly linked list

→ Step 1 :- if start = null
write underflow
Go to step 9 [END OF IF]

Step 2 :- set PTR = START

Step 3 :- Repeat step 4 while PTR → data != num

Step 4 :- set PTR = PTR → next [END OF LOOP]

Step 5 :- set TEMP = PTR → PREV

Step 6 :- set TEMP → prev → next = PTR

Step 7 :- set PTR → PREV = TEMP → PREV

Step 8 :- Free TEMP

Step 9 :- EXIT

20) deliting the node after a given node in doubly linked list

→ Step 1 :- if start = null
write underflow
Go to Step 9 [END OF IF]

Step 2 :- set PTR = START

Step 3 :- Repeat Step 4 while PTR → DATA != Num

Step 4 :- set PTR = PTR → next [END OF LOOP]

Step 5 :- set temp = PTR → next

Step 6 :- set PTR → next = temp → next

Step 7 :- set temp → next → prev = PTR

Step 8 :- Free temp

Step 9 :- EXIT.

Unit No :- 05 (Trees)

1) Pre-order Traversal :-

→ Step 1 :- Repeat steps 2 to 4 while Tree != null

Step 2 :- write tree -> data

Step 3 :- PreOrder (tree -> left)

Step 4 :- Preorder (tree -> right)

[END OF LOOP]

Step 5 :- END.

2) In order Traversal :-

→ Step 1 :- Repeat steps 2 to 4 while TREE != null

Step 2 :- Inorder (tree -> left)

Step 3 :- write tree → data

Step 4 :- inorder (tree -> right)

[END OF LOOP]

Step 5 :- EXIT

3) Post order Traversal :-

→ Step 1 : Repeat steps 2 to 4 while Tree != null

Step 2 :- Postorder (tree -> left)

Step 3 :- Postorder (tree -> right)

Step 4 :- write Tree -> data

[END OF LOOP]

Step 5 :- END.

EXPERIMENT : No. Page No. Date

4) Operations on BST :-

→ Step 1 :- if tree → data = val or tree = null RETURN Tree
ELSE
if val < tree → data
Return searchElement (tree → left, val)
else
Return searchElement (tree → right, val)
[END OF IF]
[END OF IF]

Step 2 : END

5) inserting a new node in a BST :-

→ Step 1 : if tree = null
allocate memory for tree
set tree → data = val
set tree → left = tree → right = null
else
if val < tree → data
insert (tree → left, val)
else
insert (tree → right, val)
[END OF IF] [END OF IF]

Step 2 : END.

Teacher's Sign. :

6) Deleting a Node from the BST :-

```
→ Step 1:- if tree = null
              write "val not found in the tree"
           else if val < tree → data
              delete (tree → left , val)
           else if val > tree → data
              delete (tree → right, val)
           else if tree → left and tree → right
              set temp = findLargestNode (tree → left)
              set tree → data = temp → data
              delete (tree → left, temp → data)
           else
              set temp = tree
           if tree → left = null and tree → right = null
              set tree = null
           else if tree → left ! = null
              set tree = tree → left
           else
              set tree = tree → right
           [END OF IF]
           FREE TEMP
         [END OF IF]
   Step 2: END
```

EXPERIMENT : No. Page No. Date

Unit NO: 06 (Graph)

1) Algorithm for Breadth-first Search:-

→ Step 1:- Set Status = 1 (ready state) for each node in G

Step 2:- enqueue the starting node A and it's STATUS = 2 (waiting state)

Step 3:- Repeat steps 4 and 5 until queue is empty

Step 4:- dequeue a node N. Process it's and set it's STATUS = 3

Step 5:- enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

[END OF LOOP]

Step 6:- EXIT.

Teacher's Sign. :

2) Algorithm for depth-first search:-

→ Step 1:- set status = 1 (ready state) for each node in G

Step 2:- push the starting node A on the stack and set it's status = 2 (waiting state)

Step 3:- Repeat Steps 4 and 5 until the Stack is empty.

Step 4:- Pop the top node N. Process it and set it's status = 3 (processed state)

Step 5:- Push on the stack all the neighbours of N that are in the ready state (whose status = 1) and set their status = 2 (waiting state)

[END OF LOOP]

Step 6:- EXIT