Algorithms:

a) To accept percentage from user for N number of students.

Step 1: Create an empty array 'percentages'
Step 2: for i=1 to N do
    i) Input percentage for student i.
    ii) Append the input percentage to array.
step 3: Return 'percentages'.

b) To perform bubble sort & print sorted elements.

Input: Array 'arr' containing percentage of students.

Output: sorted array in ascending order.

Step 1: for i=1 to length(arr) - 1 do
step 2: Set key = arr[i]
step 3: set j = i-1
    while j>=0 and arr[j]> key
    do

Step 4: set arr[j+1] = arr[j].

Step 5: Decrease j by 1.

Step 6: set arr[j+1] = key

Step 7: Print the sorted array 'arr'.

c) To perform selection sort and print sorted elements.

Step 1: Input arr (array of percentages)
Step 2: for i from 1 to length of arr-1:
set key = arr[i]
set j = i-1
while j >= 0 & arr[i] > key
set arr[j+1] = arr[j]
Step 3: Decrement j.
set arr[j+1] = key
Step 4: Output sorted array.

* Algorithms:

a) Accept percentage from user for N students.

1. Input N: Number of students.
2. Initialize an empty list Percentage[].
3. For i=1 to N:
   a. Prompt the user to enter the percent for student i.
   b. Append the percentage to the percentage[] list.
4. Output the percentage.

b. Perform insertion sort and print sorted elements.

1. Input: Percentage[] (list).
2. For i = 1 to length(Percentage) - 1:
   a. set key = Percentage[i]
   b. set j = i - 1
   c. while j >= 0 and percentage[j] > key
      i. Percentage[j+1] = Percentage[j]
      ii. Deerement j by 1.
   d. percentage[j+1] = key.
3. Output sorted Percentage[].

c) Perform shell sort and Print sorted elements.

1. Input : Percentage [ ] (list).
2. Set n = length (Percentage)
3. Initialize gap = n // 2
4. while gap > 0 :
   a. for i = gap to n-1 :
      i. Set temp = percentage [i]
      ii. Set j = i
      iii. While j >= gap and percentage [j-gap] > temp :
         A. Percentage [j] = Percentage [j-gap]
         B. Decrement j by gap.
      iv. Percentage [j] = temp.
   b. Reduce gap by half.
5. Output : Sorted Percentage [ ].

d) Display Top five scores of insertion sort.

i. Input : Sorted Percentage [ ]
ii. Initialize Top-five [ ] = sorted Percentage [-1 : -6 : -1].
iii. Output : Top-Five [ ].

e) Display Top five scores of shell sort.

i. Input: sorted Percentage [ ].
ii. Initialize Top-five = sorted Percentage [-1:-6:-1]
iii. Output : Top-five[ ].

Pivot : 81
[45, 72, 81, 90]

Final sorted : [09, 18, 27, 39, 45, 72, 81, 90]
array

* Algorithm:

a) Accept percentage from user for N students.

i. Input : N (Number of students)
ii. Initialize an empty an empty list.
iii. For i=1 to N:
    a. Input the percentage of students
    b. Append the entered percentage to the list.
iv). Output the list.

b) Perform quicksort and Print sorted elements.

i. Input : Percentage [ ], low (start index) high (end index).
ii. If low < high:
    a. Set pivotIndex = Partition (Percentage [ ], low, high)
    b. Recursively call quick_sort (Percentage [ ], low, pivotIndex-1)

c. Recursively call quick_sort (Percentage[], pivotIndex+1, high).

Algorithm: Partition

1. Input : Percentage[], (high, low).
2. Set pivot = Percentage [high].
3. Initialize i = low - 1.
4. For j = low to high - 1.
   a. If Percentage [j] <= pivot:
      i. Increment i by 1.
      ii. Swap percentages[i] and percentages[j].
5. Swap Percentage [i+1] and percentage [high].
6. Return i+1 (pivot index).

c) Display top five scores of quicksort.

1. ~~Iptut~~ Input : sorted Percentages[].
2. If length (sorted Percentage[]) < 5:
   a. Set Top-Five[] = sorted Percentage [-1::-1].

   Else:
   b. Set Top-Five[] = Sorted Percentage [-1:-6:-1].
3. Output : Top-five[].

Algorithms:

a) Initialize linked list:

i. Create a linked list structure with:
   a. Node (containing a pointer to the next node)
ii. Set the Head of the linked list to Null.
iii. Output: Initialized linked list.

b) Add Members (President or secretary)

i. Input: Linked_list, member_name, position.
ii. Create a node with:
   a. Data = member_name
   b. Next = NULL.
iii. If position = "President":
   a. Set new_node.next = linked_list.head.
   b. Update Linkedlist.head = new_node.
iv. Else if position = "secretary":
   a. Traverse to the last node of the linked list.
   b. Set last node.next = new_node.
v. ~~Else~~
   Output: Updated linked_list.

Page No.
Date

c) Delete members (President, Secretary).

1. Input: Linked list, Position.
2. If Linkedlist.Head = NULL:
   a. Output: "List is empty".
   b. Exit.
3. If position = "President":
   a. Set temp = Linked_list.Head
   b. Update Linked_list.Head = Linked_list.Head.Next.
   c. Delete temp.
4. Else if position = "Secretary":
   a. Traverse to the second last node.
   b. Set temp = last_node.next.
   c. Update last_node.next = NULL.
   d. Delete temp.
5. Else (General member):
   a. Traverse the linked list to find the node to delete.
   b. Update the previous node's next pointer to skip the target node.
   c. Delete the target node.
6. Output: Updated linked_list.

d) Compute total number of Members.

1. Input : Linked_list.
2. Initialize count = 0.
3. Set current = Linked_list.Head.
4. While current != NULL :
    a. Increament count by 1.
    b. Move current to current.next.
5. Output : count.

e) Display Members :

1. Input : Linked_list.
2. If Linked_list.Head = NULL :
    a. Output : "List is empty";
    b. exit.
3. Set current = linked_list.Head
4. While current != NULL:
    a. Print current.data.
    b. Move current to current.next.
5. Output : All members names.

Page No.
Date

f) Concatenate Two lists:

1. Input : Linked_list1, Linked_list2.
2. If Linked_list1.Head = NULL :
   a. Set linked_list1.Head = linked_list2. Head.
   b. Output : Linked_list1.
   c. Exit.
3. Set current = Linked_list1.Head.
4. Traverse to the last node of Linked_list1.
5. Set current.Next = Linked_list2.Head.
6. Output : Concatenated Linked_list.

9

Algorithms:

a) To display list of available seat:

1. Input: head (Pointer to the start of the list).
2. If head = NULL:
   a. Output = "No seats available".
   b. Exit.
3. Set current = head.
4. Repeat:
   a. If current.status = "available":
      i. Print current.seat_number.
   b. Set current = ~~head~~ current.next.
5. Until current = head (Complete the circular traversal).
6. Output: All available seats.

b) Book seats:

1. Input: head, seat_number_to_book.
2. If head = NULL:
   a. Output: "No seats available".
   b. Exit.
3. Set current = head.
4. Repeat:
   a. If current.seat_number = seat_number_to_book:

i. If current.status = "booked"
   A. Output: "Already booked"
   B. Exit.
ii. Else:
   A. Set current.status = "booked"
   B. Output: "Seat booked successfully."
   C. Exit.
b. Set current = current.next.
5. Until current = head
6. Output: "Seat not found".

c) Cancel seats:

1. Input: ~~seat~~ Head, seat_number_to_cancel.
2. If head = NULL:
   a. Output: "No seats available"
   b. Exit.
3. Set current = head.
4. Repeat:
   a. If current.seat_number = seat_number_to
   i). If current.seat = "available". =cancel.
      A. Output: "Seat is already available".
      B. Exit.
   ii). Else.
      A. ~~Set~~ current.status = "available"
      B. Output: "Seat ~~booking~~ booking canceled succesfully".
      C. Exit.

Page No.
Date

Flowcharts and Algorithms:

a) To display free slots.

1. for each timeslot from starttime to endtime:
2. SET booked = false.
3. SET temp = head.
4. WHILE temp IS NOT NULL:
5. IF temp.timeslot == timeSlot.
6. SET booked = true.
7. BREAK
8. END IF
9. SET temp = temp.next.
10. END WHILE
11. IF booked == false:
12. PRINT timeSlot as free:
13. END IF
14. END FOR.

b) Book Appointment.

1. SET temp = head
2. WHILE temp IS NOT NULL:
3. IF temp.timeslot == timeslot:
4. PRINT "slot already booked"
5. RETURN
6. END IF.
7. SET temp = temp.next.

8. END WHILE
9. CREATE a new slot node with timeslot and description.
10. INSERT the new node at the beginning of the list.
11. PRINT "Appointment booked".

c) Cancel Appointment:

1. SET time = head.
2. SET Prev = NULL
3. ~~NULL~~ While temp IS NOT NULL:
4. If temp.timeslop = timeslop.
5. IF prev IS NOT NULL:
6. SET prev.next = temp.next
7. ELSE:
8. SET head = temp.next.
9. DELETE temp
10. PRINT "Appointment canceled"
11. RETURN
12. END IF
13. SET prev = temp
14. SET temp = temp.next
15. END WHILE
16. PRINT "No appointment found to cancel".

Page No.
Date
Dr.D.Y.PATIL Technical Campus

* Algorithms :

a) Check empty stack:

```
1. If stack.size == 0
2.     RETURN True
3. Else
4.     RETURN False.
```

b) Check full stack.

```
1. IF stack.size == capacity
2.     RETURN True
3. ELSE :
4.     RETURN False
```

c) Initialize stack :

```
1. SET stack.size = 0
2. SET stack.element = empty array or
                       linked list.
```

d) Push element to stack.

```
Algorithm Push (stack, element)
1. IF ISFULL (stack, capacity)
2.     PRINT " stack overflow"
3. Else :
4.     stack.element [stack.size] = element.
```

5. INCREMENT stack.size

e) Pop elements from stack.

Algorithm Pop(stack):

1. IF ISEMPTY(stack):
2. PRINT "Stack Underflow."
3. RETURN NULL:
4. ELSE:
6. SET element = stack.elements[stack.size
7. - 1]
7. DECREMENT stack.size
8. RETURN element.

f) Display stack as well ~~per~~formed Paranthesis expression.

1. IF ISEMPTY(stack):
2. PRINT: "Expression is well formed."
3. Else:
4. PRINT: "Expression is not well formed. Remaining elements in stack:"
5. FOR each element in stack.elements:
6. PRINT element.

Algorithms:

1) Infix to Postfix conversion.

Input: An Infix expression
Output: A postfix expression

1. Initialize an empty stack for operaters and an empty string for the postfix expression.
2. for each character in the infix expression:
   - If the character is not operand, append it to the postfix string.
   - If the character in an operater:
     a. while the stack is not empty & the precedence of the top operator is greater than or equal to the current operater:
        i. Pop the operater from the stack and append it to the postfix string.
     b. Push the current operator into the stack.
   - If the character is (, push it into the stack.
   - If the character is), Pop and append operators from the stack

until ')' is encountered, then discard (.
3. After traversing the expression, pop all remaining operators in the stack and append them to the postfix string.
4. Return the postfix expression.

2) Algorithm for evaluating postfix expression.

Input: A postfix expression
Output: The evaluated result.

1. Initialize an empty stack for operands
2. For each character in the postfix expression:
   i. If the character is a operand, push it into the stack.
   ii. If the operater is a operator:
      a. Pop the top two elements from the stack.
      b. Perform the operation with the second popped element as the left operand and the first was the right operand.
      c. Push the result back into the stack.
3. After traversing the expression, the result will be the top of stack.

Algorithms :

a) Check if queue is Empty :

```
1. If front == -1 OR front > rear :
2.     RETURN True
3. ELSE :
4.     RETURN False.
```

b) Check if queue is full :

```
1. If rear == capacity - 1 :
2.     RETURN True
3. ELSE :
4.     RETURN False.
```

c) Insert Element in Queue :

```
1. If ISFULL (queue, capacity) :
2.     PRINT "Queue Overflow"
3. ELSE :
4.     If front == -1 :
6.         SET front = 0
8.     INCREMENT rear
7.     queue[rear] = element
8.     PRINT "Job added successfully".
```

d) Delete element from Queue:

```
1. If   IS EMPTY (queue):
2.          PRINT " Queue underflow"
3. Else :
4.          PRINT " Job removed:", queue[front]
5.          INCREMENT front
6.          IF front > rear :
7.                  SET front = rear = -1 .
```

e) Display Elements of Queue.

```
1. IF IS EMPTY (queue):
2.          PRINT " Queue is empty"
3. ELSE :
4.          FOR i FROM front TO rear :
5.          PRINT queue[i]
```

Page No.
Date

Algorithms :

1) Check if the queue is empty.

```
1. IF front == -1 :
2.     RETURN True
3. ELSE
4.     RETURN False
```

2) Check if the Queue is full :

```
1. If (front == 0 AND rear == capacity-1)
2.    OR (front == rear+1):
3.    RETURN True
4. ELSE
5.    RETURN False.
```

3) Insert element in queue from rear :

```
1. IF ISFULL() :
2.     PRINT " Queue overflow"
3. ELSE
4.     If front == -1 :
5.         SET front = 0
6.     INCREMENT rear (circularly) : rear =
           (rear+1) % capacity.
7.  PRINT queue[rear] = element
8.  PRINT " Element added at rear ".
```

d) Insert Element in Queue from front.

```
1. IF ISFULL():
2.     PRINT "Queue overflow"
3. ELSE:
4.     IF front == -1:
5.         SET front = rear = 0
6.     ELSE:
7.         DECREMENT front (circularly):
           front = (front - 1 + capacity) % capacity
8. queue[front] = element.
9. PRINT "Element added at front"
```

e) Delete front element from Queue.

```
1. IF ISEMPTY():
2.     PRINT "Queue underflow"
3. ELSE:
       PRINT "Delete element:", queue[front]
       IF front == rear:
           SET front = rear = -1
       ELSE:
           INCREMENT front (circularly):
           front = (front + 1) % capacity.
```

f) Delete Rear element from queue:

```
1. IF ISEMPTY():
2.     PRINT "Queue Underflow"
3. ELSE:
       PRINT "Deleted element :", queue[rear]
       IF front == rear:
           SET front = rear = -1
       ELSE:
           DECREMENT rear (circularly):
           rear = (rear - 1 + capacity) %.
                  capacity.
```

g) Display Elements of queue:

```
1. If ISEMPTY():
       PRINT "Queue elements"
       SET i = front.
       DO:
           PRINT queue[i]
           i = (i+1) %. capacity.
       WHILE i != (rear+1) %. capacity.
```

Page No.
Date

Algorithms:

a. Check if the queue is empty.

```
1. If front == -1:
2.     PRINT True
3. EISE:
4.     PRINT False
```

b. Check if the queue is full:

```
1. If (rear+1) % capacity == front:
2.     RETURN True
3. EISE:
4.     RETURN False
```

c. Insert element in queue from Rear:

```
1. If ISFULL():
2.     PRINT "Queue overflow"
3. EISE:
4.     IF front == -1:
5.         SET front = 0.
6.     rear = (rear+1) % capacity.
7.     queue[rear] = element
8.     PRINT "Element added"
9. END
```

a) Delete front Element from queue:

```
1. IF ISEMPTY:
2.     PRINT "Queue underflow"
3. ELSE:
4.     PRINT "Deleted element:", queue[front]
5.     IF front == rear:
6.         SET front = rear = -1
7.     Else:
8.         front = (front + 1) % capacity.
```