Data structures and Algorithms

①

AIM: Program to find the largest and smallest numbers in an unsorted array

Algorithm:

1. Input the array elements.
2. initialize small = large = arr[0]
3. Repeat from i = 2 to n
4. if (arr[i] > large)
5. large = arr[i]
6. if (arr[i] < small)
7. small = arr[i]
8. print small and large.

(2)

Aim: Write a program on insertion sort

Algorithm:

1. if it is the first element, it is already sorted. return 1;
2. Pick next element
3. Compare with all elements in the sorted sub-list
4. Shift all the elements in the sorted sub-list that is greater than the value to be stored
5. Insert the value
6. Repeat until list is sorted

(3)

Aim : Write a program on Quick sort

Algorithm :

1. choose the highest index value has pivot
2. Take two variables to point left and right of the list excluding pivot
3. left points to the low index
4. right points to the high index
5. while value at left is less than pivot move right
6. while value at Right is greater than pivot move left
7. if both step 5 and step 6 does not match swap left and right
8. if left ≥ right, the point where they met is new pivot

④

Aim : Write a program on Merge sort.

Algorithm :

1. if it is only one element in the list it is already sorted, return.
2. divide the list recursively into two halves until it can no more be divided.
3. Merge the smaller lists into new list in sorted order.

⑤

Aim : Create a program on singly linked list

Algorithm :

1. if PTR = Null

   write overflow

   Go to step 7

   (End of if)

2. Set New_Node := PTR

3. Set PTR = PTR -> Next

4. Set New_Node -> Data = Val

5. Set New_Node -> Next = Head

6. set Head = New_Node

7. Exit

⑥

Aim : Creat a program on Doubly linked list

Algorithm :

i) insert at Beginning

1. start
2. input the Data to be inserted
3. Create a new node
4. New Node → Data = Data . New Node → Lpoint = Null
5. if start is Null new node → R point = Null
6. Else Newnode → Rpoint = start.
   start → L point = New Node
7. start = New Node
8. stop

ii) insert at End

1. start
2. input DATA to be inserted
3. Create a NewNode
4. NewNode → DATA = DATA
5. NewNode → R Point = Null
6. if (start equal to Null)
7. Stop

iii) insertion at location:

1. start
2. input the DATA and POS
3. initialize Temp = start ; i=0
4. Repeat the step 4 if (i less than POS) and (Temp is not equal to Null)
5. Temp = Temp → R Point ; i = i+1
6. if (Temp not equal to Null) and (i equal to POS)
7. Creat a New Node
8. New Node → Data = Data
   New Node → R point = Temp → R point
   New Node → L point = Temp
9. (Temp → R Point) → L Point = New Node
   Temp → R point = New Node
10. Else
    Display "Position Not found"
11. STOP.

(7)

Aim : Write a program on stack operations using linked list

Algorithm :

1. Push() operation

1. Create a node new and declare variable top
2. Set new data part to be Null
3. Read the node to be inserted
4. check if the node is Null, then print "memory is Full"
5. if node is not Null, assign the item to data part of new and assign top to link part of new and also point stack head to new

2. POP()

1. check if the top is Null, then print "stack underflow"
2. if top is not Null, assign the top's link part to ptr and assign ptr to stack_head's link part

3. Peek()

1. Start
2. print or store the node pointed by top variable
3. Stop

(8)

Aim: Write a program on Queue using Arrays

Algorithm:

1. Enqueue()

1. check whether queue is full (rear == size - 1)
2. if it is full, then display "Queue is Full" and terminate the function
3. if it is Not full, then increment rear value by one (rear++) and set queue[rear] = value.

2. Dequeue()

1. check whether queue is Empty (front == rear)
2. if it is Empty, then display "Queue is Empty"
3. if it is not Empty, then increment the front value by one (front ++). Then display queue(front) as a deleted element. Then check whether both front and rear are equal (front == rear), if it True, then set both front and rear to '-1' (front = rear = -1)

3. display ( )

1. check whether queue is Empty (front == rear)
2. if it is Empty, then display "Queue is Empty"
3. if it is NOT empty, then define an integer variable 'i' and set 'i = front + 1'.
4. Display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i' value reaches to rear (i <= rear)

(9)

Aim : Write a program on Queue operations using linked list

Algorithm :

1. Enqueue ()

1. Create a newnode with given value and set 'new Node → next' to Null
2. check whether queue is Empty (rear == Null)
3. if it is Empty then, set front = new Node and rear = new Node
4. if it is NOT empty then, set rear → next = new Node and rear = newNode

2. De queue ()

1. check whether queue is Empty (front == Null)
2. if it is Empty, then display " Queue is Empty"
3. if it is NOT empty then, define a Node pointer 'temp' and set it to 'front'
4. then set 'front = front → Next' and delete 'temp' (free(temp))

3. display()

1. check whether queue is Empty (front == Null)
2. if it is NOT Empty then, display "Queue is Empty"
3. if it is not Empty then, define a Node pointer 'temp' and initialize with front.
4. Display 'temp → data --> ' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → Next != Null).
5. finally; Display 'temp → data --> Null'.