Program No:1

Aim: Merge two sorted arrays in a third array

Algorithm:-

MERGING (array1, array2, merge, n, m)

Let array1 and array2 be sorted arrays with m and n elements, respectively. The algorithm merges array1 and array2 into an array merge with m+n elements.

1. SET i=0 j=0, k=0 //[initialization]
2. Repeat while i<m and j<n //[Compare]
   if array1[i] < array2[j] then
   SET merge[k] = array1[i]
   SET k = k+1 and i = i+1
   Else
   SET merge[k] = array2[j]
   SET k = k+1 and j = j+1
   end of if
   End of loop.
3. Repeat while i<m, then:
   SET merge[k] = array1[i]
   SET i = i+1

SET k = k+1

End of loop.

4. Repeat while j<n, then

SET merge [k] = array [j]

SET j = j+1 and k = k+1

End of loop

3. Exit

Program No: 2.

Singly Linked Stack.

Aim: Singly linked stack - push, pop, linear search

Algorithm:

1. Start.
2. If user slect push opration then.
3. ~~Start~~ Create a newNode with the given data.
4. If top == NULL then,
   (check whether stack is empty or not)
5. SET top = newNode
   SET newNode → next = NULL.
   else
   SET newNode → next = top
   top = newNode.
   End of if
6. If user select pop operation then.
7. If top == NULL then:
   (Check whether stack is empty)
   display "stack is empty"

else :

SET temp = top

(Create a temporary node and set it to top)

display item → data.

8. SET top = temp → next

(make top point to the next node)

9. free (temp)

(Delete the temporary node)

10. If user select search operation then

11. ~~Delete~~ Declare a pointer variable temp and the variable key that holds the value to be searched.

12. SET temp = top
SET flag = 0

13. Repeat while temp != NULL
if temp → data == key then
display "element found"
SET flag = 1
End of if structure
go to step 14.
Else
SET temp = temp → next
End of while.

14 : If flag == 0 then.
display "element not found"
End of if structure.

15 : If user select display operation

16 : declare a pointer node ptr.
SET node ptr = top

17 : If nodeptr == NULL then
display "stack is empty"
End of if structure"

18 : While nodeptr != NULL then
print nodeptr → data
SET nodeptr = nodeptr → next

19 : If node ptr != NULL then
print "- → "

End of if
End of while

20 : Exit

Program No: 3.

Circular Queue.

Aim :

Algorithm :

1. Start
2. If user select the insertion operation then
3. Declare a variable item with some value.
4. If front == 0 && rear == size-1
   // front = rear + 1 then
   display queue overflow"
   End of if structure
5. If front == -1 then
   SET front = 0
   SET rear = 0
   End of if
6. if rear == size-1
   SET rear = 0
   else
   SET rear = rear+1

End of if

7: SET item = queue[rear]

8: If user select deletion operation then,

9: If front == -1 then
display "Queue underflow"
End of if structure.

10: If front == rear then
SET front = -1
SET ~~front~~ rear = -1
End of if

11. If front == size - 1 then
SET front = 0
else
SET front = front + 1
End of if

12: If user select the display operation then

13: SET front_pos = front
SET rear_pos = rear.

```
14 : If front == -1 then
     Display "Queue is empty"
     End of if
15 : If front_pos <= rear_pos then
     while front_pos <= rear_pos then
     print cq [front_pos]
     SET front_pos = front_pos + 1
     End of while
     else
     Repeat while front_pos <= size - 1
     print cq [front_pos]
     SET front_pos = front_pos + 1
     End of while
16 : SET front_pos = 0
17 : Repeat while front_pos <= rear_pos
     then
          cq [front_pos]
     Set front_pos = front_pos + 1
     End of while
     End of if
```

18: If user select search operation then

19: Declare a variable ser with value to be searched.

20: Declare a temporary variable temp then

SET temp = ser

21: SET i = font

22: Repeat for i<= rear then;

If t == &cq [i] then

print i+1

SET j = j+1

End of if

If j == 0 then

Display "item not found".

End of if

SET i = i+1

End of for

23: Exit

Program No: 4

Aim:

Algorithm:

1: Start
2: If user start the union operation then:
3: Declare two array set1[i] and set2[i], Declare two variables n_1, n_2 for holding the size of two arrays.
4: Read elements into the arrays set1[i] and set2[i]
5: if $n_1 == n_2$ then
6: SET i=0
7. Repeat for $i < n_2$ then
SET set3[i] = set1[i] || set2[i]
8: SET i = i+1

9: SET i = 0

10: Repeat for i < 2 then

print set3[i]

11: SET i = i+1

End of for

End of if

else

print "size are not equal"

exit

12: If user select intersection operation then.

13: ~~Dc~~ Declare two array set1 [i] and set2 [i] with size n_1, n_2 Respectively and Read elements to the arrays.

14: if $n_1 == n_2$ then

15: SET i = 0

16: Repeat for i < n_2 then

17: SET set3 [i] = set1[i] && set2 [i]

18: SET i = i+1

19 : SET i = 0

20 : Repeat for i < n2 then;
print set3[i]

21 : set i = i+1

End of for

End of if

Else
print "sets are not equal"

exit

22. If user select the substraction
then

23. Declare two array set1[i] and
set2[i] with n_1, n_2 size respectively
and input the elements to the array.

24 : if $n_1 == n_2$ then

25 : set i = 0

26 : Repeat for i < n2 then :

~~SET~~
SET set3[i] = set1[i] - set2[i]

27 : i = i+1
End of for loop

```
28 : x=4 i=0
29 : Repeat for i<n2 then
        print str3[i]
30 : str i = i+1
        End of for loop
        End of if
     else
        "print size are not equal"
31 : Exit
```

Program No: 5.

Binary Search Tree.

Aim:

Algorithm

1: Start

2: If user select the insertion's operation then

3: Create a new BST node and assign values to it.

4: Create tree (node, data) // call the createtree function then with the root value and the data entered by user

5: If root == NULL then;

6: Declare a temporary variable temp

SET temp → data = data

SET temp → left → right = NULL;

return the new node temp to the calling function

End of if

7: If data < (node → data)

8: ~~If data~~ Call the create node function with node → left

9: If data > node → data

10: Call the create tree (node → left, data)

End of if

and assign the return value in node → left

node → left = Create tree (node → left, data)

End of if

9: Else if data > node → data.

10: Call the create tree function with node → right and assign the return value in node → right.

node → right = create tree (node → right, data)

End of if

11: return the original root pointer 'node' to the calling function

12: If the user select the search element operation then.

13: Search (root, data) // call the search function with root value and the element ~~the~~ that to be searched.

14: if root == NULL
print "element not found"
end of if

15: If data < root → data then:
Call the search function with root → left and assign the return value is root → left
root → left = search (root → left, data)
end of if

16: If data → root → data then,
call search function with root → right and assign the return value is root → right
root → right = search (root → right, data)

End of if

else
print "Element found is" root → data.

17: Return the original root pointer node to the calling function.

18: If the user select the deletion operation then:

19: del (node, data) // call the deffunction with root value and the element to be deleted.

20: declare a temporary variable temp

21: If node == NULL then
point "element not found"
end of if

22: If data < node → data, then: Call the del function with node → left and assign the return value to node → left.
node → left = del (node → left, data)
end of if.

23: if data > node → data then call the del function with node → right and assign the return value to node → right
End of if

24 : Else . // delete this node and replace
25 with either minimum element in the right subtree or maximum element in the left subtree.

25 : If node → right && node → left
// replace with minimum element in the right subtree.

26: Call find min function with node → right then return value assign in temp
Go to step 32.

SET temp = findmin (node → right)
SET node → data = temp → data
// replaced it with some other node.

27: Call function del with value node → right, temp → data and return value assign in node → right

else

28 : SET temp = node.
// If there is only one or zero children then we can directly remove it from the tree and connect.

its parent to its child.

29: If node → left == NULL then
    SET node = node → right
    else
30: if node → right == NULL then
    SET node = node → left
31: free (temp)
    End of if
    End of if
    End of if
32: find min (node)
33 - If node == NULL then.
    return NULL
    Go to step 26.
    End of if
34: If node → left then
    Call the function find min
    with value (node → left) then

return the value to calling function
return function (node → left)
Else
return node
Goto step 24.
End of if

35 : If the user select the display option
then,

36 : inorder (node)
Call the inorder function with root
value

37 : If node != NULL then
inorder (node → left)
call the function inorder with value
node → left

38 : Print node → data
Inorder (node → right)
Call the function inorder with
value node → right
End of if

39 : Exit

Program No:6.

Doubly Linked List

Aim: Doubly linked list - insertion, deletion search.

Algorithm:

1. Start
2. If user select the insert operation at begining then:
3. If head == NULL then:
4. Perform step 56 to 59.
   [call the function create ()]
5. SET head = temp
   temp = head
6. Else
7. perform step 56 to 59.
8. SET temp → next = head
   SET head → prev = temp
   SET head = temp
9. End of If

9: If user choose the operation insert node at the end then

10: If Head == NULL then:

11- perform Step 56 to 59.

12. SET Head = temp

SET temp 1 = Head

13- Else.

14. perform Step 56. to 59.

15. SET temp 1 → next = temp

temp → prev = temp 1

temp 1 = temp

End of if

16: If user choose insert at any position then:

17 Read the position and store it in to the variable POS.

18: SET temp2 = Head

19: If pos < 1 || pos >= count +1 then:

Display "position is out of range to insert"

Exit.

End of if.

20: If Lead == NULL && pos != 1 then:

Display "Empty list cannot insert other than 1st position"

Exit

End of if

21. If Lead == NULL && pos == 1 then:

22. perform step 56 to 59.

(Call function create())

23: SET Lead = temp

SET temp1 = Lead

[End of if]

Exit

24: Else

Repeat

25: While i < pos then:

26: SET temp2 = temp2 → next

SET i = i + 1

End of while.

27: Perform step 56 to 59.

28: Set temp → prev = temp2.
Set temp → next = temp2 → next
Set temp2 → next → prev = temp
Set temp2 → next temp

29: If user choose the operation deletion then:

30: Read the position then it store in to the variable pos.

31: Set temp2 = head

32: If pos < 1 || if pos >= count + 1 then:
Display "position out of range to delete"
End of if
Exit

33: If head == NULL then
Display "Empty list no elements to delete".
End of if
Exit

34: Else

35: Repeat while i < pos

Set temp2 = temp2 → next

Set i = i+1

End of while

36. If i == 1 then:

If temp2 → next == NULL then:

Display "Node deleted from list"

37: free temp2.

SET temp2 = head = NULL

Exit

38: If i != 1 then

temp2 → prev → next = temp2 → next

39: if i == 1 then

SET head = temp2 → next

Display "Node deleted"

40. free temp2.

41: SET count = count - 1

42: If user select display operation then:

43: Set temp 2 = head

44: If temp 2 = = NULL

Display "List empty to display"

Exit

45: while temp 2 → next 1 = NULL then:

print temp 2 → n.

SET temp 2 = temp 2 → next

46: print temp 2 → n.

47: If user select search operation then:

48: SET temp 2 = head.

49: If temp 2 = = NULL then

Display "List empty to search for data".

Exit

50. Read the value to be searched and store it into the variable data.

51: While temp2 != NULL then:

52: If temp2 → n == data then:
print count + 1
Exit

53: Else
SET temp2 = temp2 → next
SET count = count + 1
End of if
of while

54: Display "not found"

55. Exit

56: Set temp → prev = NULL

57: SET temp → next = NULL

Display "Enter values to node"

58: Read data and assign it into
temp → n
temp → n = data

59: Count = count + 1

60: Exit

Program No: 7

Disjoint_set.

Aim: Disjoint sets and the associated operations (create, union, find).

Program:

1. start
2. Read the number of elements from user and store it in to the dis.n
3. Call function makeset() then:
4. SET i=0
5. Repeat for i<dis.n then:

   SET dis.parent[i] = i

   SET dis.rank[i] = 0

   SET i = i+1.

   [End of for loop]

6. User select the union operation then:
7. Read the elements to perform union and store in to x and y respectively.

8. // perform find operation with x and y store result in to xset and yset perform step 23.

9. If xset = yset then:
   [End of if]

10. If dis.rank[xset] < dis.rank[yset]
    then:
    SET dis.parent[xset] = yset.
    SET dis.rank[xset] = 1.
    End of if

11. Else if dis.rank[xset] > dis.rank[yset] then:
    SET dis.parent[yset] = xset;
    SET dis.rank[yset] = -1
    End of if

12. else

13. SET dis.parent[yset] = xset
    SET dis.rank[xset] = dis.rank[xset] + 1.
    SET dis.rank[yset] = -1

14. If user choose find operation then:
15. Read the elements to check if and store the value into the variables x and y respectively
16. If find x == find y then:
display "connected components"
17. else.
18. Display "not connected components"
18: If user select the display operation then
19. SET i=0
20. Repeat for i < dis.n then
print dis.parent[i]
SET i = i+1
End of for loop
21. SET i=0
22. Repeat for i < dis.n then:
print dis.rank[i]

SET i = i + 1
End of for loop

23. If dis.parent [x] != x
Then
SET dis.parent [x] = find (dis.parent [x])
return dis.parent [x]

24. End