Program-1

Aim:- To merge two sorted arrays.

Algorithm:-

Step 1 : Start

Step 2 : Declare the variables.

Step 3 : Read the size of first array.

Step 4 : Read elements of first array in sorted order.

Step 5 : Read the size of second array.

Step 6 : Read the elements of second array in sorted order.

Step 7 : Repeat step 8 and 9 while $i<m$ & $j<n$

Step 8 : check if $a[i] >= b[j]$ then $c[k++]=b[j++]$

Step 9 : else $c[k++]=a[j++]$

Step 10 : Repeat step 11 while $i<m$

Step 11 : $c[k++]=a[j++]$

Step 12 : Repeat step 13 while $j<n$

Step 13 : $c[k++]=b[j++]$

Step 14 : Print the first array

Step 15 : Print the second array

Step 16 : print the merged array

Step 17 : Stop.

Output

```
Size of first array
2
Enter value in sorted order
5
17
Size of second array
2

Enter value in sorted order
9
10

Array A:
5
17

Array B:
9
10

merged array
5
9
10
17
```

Program : 2

Aim : To perform stack operations.

Algorithm :

Step 1 : Start

Step 2 : Declare the node and the required variables

Step 3 : Declare the functions for push, pop, display and search an element

Step 4 : Read the choice from the user.

Step 5 : If the user choose to push an element then read the element to be pushed and call the function to push the element by passing the value to the function.

Step 5.1 : declare the new node and allocate memory for the new node.

Step 5.2 : Set new node -> data = value

Step 5.3 : check if top == null then set newnode -> next = null

Step 5.4 : Set new node -> next = top

Step 5.5 : Set top = new node and then print insertion is successfull.

Step 6 : If user choose to pop an element from the stack then call the function to pop the element.

Step 6.1 : Check if top == Null then print stack is empty.

Step 6.2 : else declare a pointer variable temp and initialize it to top.

Step 6.3 : print the element that being deleted.

Step 6.4 : set temp = temp → next.

Step 6.5 : free the temp

Step 7 : If the user choose the display then call the function to display the element in the stack

Step 7.1 : Check if top = Null then print Stack is empty.

Step 7.2 : else declare a pointer variable temp and initialize it to top.

Step 7.3 : Repeat steps below while temp → next != null.

Step 7.4 : print temp → data.

Step 7.5 : step temp = temp → next.

Step 8 : If the user choose to search an element from the stack then call the function to search an element:

Step 8.1 : Declare a pointer variable ptr and other necessary variable.

Step 8.2 : Initialize ptr = top.

Step 8.3 : check if ptr = null then print stack empty.

Step 8.4 : else Read the element to be searched.

Step 8.5 : Repeat step 8.6 to 8.8 while ptr != null.

Step 8.6 : check if ptr -> data == item then print element founded and to be located and set flag = 1.

Step 8.8 = check if flag = 0 then print the element not found.

Step 9 = stop.

output

Menu

1. Push
2. Pop
3. display
4. search
5. Exit

Enter the choice : 1
Enter the element to be inserted : 2
insertion is successfull

Menu

1. Push
2. Pop
3. display
4. search
5. Exit

Enter the choice : 1
Enter the element to be inserted : 4
insertion is successful.

Menu

1. push
2. pop
3. display
4. search
5. Exit

Enter the choice : 1

Enter the element to be inserted : 10

Insertion is successfull.

Menu

1. push
2. pop
3. display
4. Search
5. Exit

Enter the choice : 2

Element deleted : 10

Menu

1. push
2. pop
3. display
4. search
5. Exit

Enter the choice : 3

4 → 2 → null

Program: 3

Aim: To perform circular queue operations.

Algorithm:

Step 1 : Start

Step 2 : Declare the queue and the other variables.

Step 3 : Declare the functions for enqueue, dequeue search and display.

Step 4 : Read the choice from the user.

Step 5 : If the user choose the choice enqueue then read the element to be inserted from the user and call the enqueue function by passing the value.

Step 5.1 : check if front == -1 and rear == -1 then set front = 0, rear = 0 and set queue [rear] = element.

Step 5.2 : else if rear + 1 % max == front or front = rear + 1 then print queue is overflow.

Step 5.3 : else set rear = rear + 1 % max and set queue [rear] = element

Step 6 : If the user choice is the option deque then call the function dequeue.

Step 6.1 : check if front == -1 and rear == -1 then print queue is underflow.

Step 6.2 : else check if front == rear then print the element is to be deleted then set front = -1 and rear = -1.

Step 6.3 : else print the element to be dequeued. set front = front + 1% max.

Step 7 : if the user choice is to display the queue then call the function display.

step 7.1 : check if front = -1 and rear = -1 then print queue is empty.

Step 7.2 : Else repeat the step 7.3 while i<= rear.

Step 7.3 : print queue[i] and set i = i % max

Step 8 : If the user choose the search then call the function to search an element in the queue

Step 8.1 : Read the element to be searched in the queue.

Step 8.2 : check if item == queue[i] then print item found and its position and increment by 1.

Step 8.3 : check if c == 0 then print item not found.

Step 9 : stop.

output

Menu
1. Insert
2. Delet
3. display
4. search
5. Exit

Enter the choice : 1
Enter the number to insert : 10

Menu
1. Insert
2. Delet
3. display
4. search
5. Exit

Enter the choice : 1
Enter the number to insert : 20

Menu
1. Insert
2. Delet
3. display
4. search
5. Exit

Enter the choice : 1
Enter the number to insert : 30

Menu
1. insert
2. Delet

3. duplay
4. Seach
5. Exit
Enter the choice : 3
10, 20, 30

Menu
1. Insat
2. Delet
3. display
4. Search
5. Exit
Enter the choice : 4
Enter the element which is to be Search : 30
Item found at location 3

Menu
1. insert
2. Delet
3. display
4. search
5. Exit
Enter the choice : 2
10 was deleted

Menu
1. Insert
2. Delets
3. display
4. Search
5. Exit

Enter the choice : 5

Program: 4

Aim : To perform doubly linked list operations.

Algorithm:

Step 1 : Start

Step 2 : Declare a structure and related variables.

Step 3 : Declare functions to create a node, insert a node in the begining. at the end and given position, display the list and search an element in the list.

Step 4 : Define function to create a node, declare the required variables.

Step 4.1 : Set memory allocated to the node = temp then set temp → prev = null and temp → next = null.

Step 4.2 : Read the value to be inseeted to the node.

Step 4.3 : Set temp → n = data and increment count by 1.

Step 5 : Read the choice from the user to perform different operation on the list.

Step 6 : If the user choose to perform insertion operation at the beginning then call the function to perform the insertion.

Step 6.1 : check if head = = null then call the function to creat a node, perform step 4 to

Step 6.2 : set head = temp and temp1 = head.

Step 6.3 : Else call the function to creat a node, perform step4 to 4.3 then set temp → next = head, set head → prev = temp and head = temp

Step 7 : If the user choice is to perform insertion at the end of the list, then call the function to perform the insertion at the end.

Step 7.1 : check if head == null then call the function to creat a new node then set temp = head and then set head = temp1.

Step 7.2 : Else call the function to create a new node then set temp1 → next = temp. temp → prev = temp1 and temp1 = temp.

Step 8 : If the user choose to perform insertion in the list at any position then call the function to perform the insertion operation.

Step 8.1 : Declare the necessary variables.

Step 8.2 : Read the position where the node need to the inserted, set temp2 = head.

Step 8.3 : check if pos < 1 or pos >= count + 1 then print the position is out of range.

Step 8.4 : checks if head == null and pos = 1 then print "empty list cannot insert other then first position"

Step 8.5 : check if head == null and pos = 1 then call the function to creat new node the set temp = head and head = temp1.

Step 8.6 : while i < pos then set temp 2 = temp2 → next then increment i by 1.

Step 8.7 : call the function to creat a new node and then set temp → prev = temp2. temp → next = temp2 → next → prev = temp. temp2 → next = temp.

Step 9 : if the user choose to perform deletion operation is the list than all the function to perform the deletion operation.

Step 9.1 : Declare the necessary variables.

Step 9.2 : Read the position where node need to be delet set temp 2 = head.

Step 9.3 : check if head == null then print the list is empty.

Step 9.4 : check if head == null then print the list is empty.

Step 9.5 : while i < pos then temp 2 = temp2 → next and increment i by 1.

Step 9.6 : check if i == 1 then check if temp2 → next == null then print node deleted free (temp2) set temp2 → head = null.

Step 9.7 : check if temp2 → next == null then temp2 → prev → next = null then free (temp2) then print node deleted.

Step 9.8 : $temp_2$ → next → prev = $temp_2$ → next then check if i != 1 then temp2 → prev → next = temp2 → next.

Step 9.9 : check if i == 1 then head = $temp_2$ → next then print node deleted then free temp2 and decrement count by 1.

Step 10 : If the user choose to perform the display operation then call the function to display the list.

Step 10.1 : Set temp2 = h

Step 10.2 : check if temp2 = null then print list is empty.

Step 10.30 : while temp2 → next, = null then print temp2 → n then temp2 = temp2 → next

Step 11 : If the user choose to perform the search operation then call the function to perform search operations.

Step 11.1 : Declare the necessary variables.

Step 11.2 : Set temp2 = head

Step 11.3 : check if temp2 == null the print the list is empty

Step 11.4 : Read the value to be searched.

Step 11.5 : while temp2 != null then check if temp2->n == data then print element found at position count+1.

Step 11.6 : Else set temp2 = temp2->next and increment count by 1.

Step 11.7 : print element not fount in the list.

Step 12 : End.

output

1. Insert at begining
2. Insert at end
3. Insert at position
4. Delet
5. Display
6. Search
7. Exit

Enter choice: 1
Enter the value to node: 5

1. Insert at begining
2. Insert at end
3. Insert at position
4. Delete
5. Display
6. Search
7. Exit

Enter choice: 1
Enter The value to node: 10

1. To Insert at beginning
2. Insert at end
3. Insert at position
4. Delete
5. Display
6. Search
7. Exit

Enter choice: 2
Enter value to node: 2.

1. insert at begining
2. Insert at end
3. Insert at position
4. Delet
5. display
6. Seach
7. Exit
Enter choice : 3
Enter the position to be inserted : 2
Enter the value to node : 13.

1. insert at begining
2. insert at end
3. insert at position
4. Delete
5. display
6. Search
7. Exit

Enter choice : 4
Enter the position to be deleted : 2
node deleted

1. insert at beginning
2. insert at end
3. insert at position
4. Delete
5. display
6. search
7. Exit

Enter choice : 6

Enter value to search : 10
Element found in 1 position

1. Insert at beginning
2. Insert at end
3. Insert at position
4. Delete
5. Display
6. Search
7. Exit

Enter the choice : 7

Program : 5

Aim : To perform set operations.

Algorithm :

Step 1 : Start

Step 2 : Declare the necessary variable.

Step 3 : Read the choice from the user to perform set operation.

Step 4 : If the user choose to perform union

Step 4.1 : Read the cardinality of 2 sets.

Step 4.2 : Check if $m_1 = n$ then Print cannot perform union.

Step 4.3 : Check else read the elements in both the sets.

Step 4.4 : Repeat the step 4.5 to 4.7 until $i < m$.

Step 4.5 : C [i] = A [i] B [i]

Step 4.6 : Print C [i]

Step 4.7 : Increment i by 1

Step 5 : Read the choice from the user to perform Intersection.

Step 5.1 : Read the cardinality of 2 sets.

Step 5.2 : Check if $m_1 = n$ then print cannot perform Inter section.

Step 5.3 : Else read the element in both the sets.

Step 5.4 : Repeat the step 5.5 to 5.7 until $i < m$.

Step 5.5 : C[i] = A[i] & B[i]

Step 5.6 : Print C[i]

Step 5.7 : Increment i by 1

Step 6 : If the user choose to perform set difference operation

Step 6.1 : Read the cardinality of a set.

Step 6.2 : check if $m_1 : n$ then print cannot Perform set difference operation.

Step 6.3 : Else read the element in both sets.

Step 6.4 : Repeat the step 6.5 to 6.8 until $i<n$.

Step 6.5 : Check if A[i] == 0 then C[i] = 0

Step 6.6 : Else if B[i] == 1 the C[i] = 0

Step 6.7 : Else C[i] = 1.

Step 6.8 : Increment i by 1.

Step 7 : Repeat Step 7.1 and 7.2 until $i<m$.

Step 7.1 : print C[i]

Step 7.2 : Increment i by 1.

Output

```
Press 1 for Union
Press 2 for Intersection
Press 3 for Subtraction
Press 4 for exit

Enter choice 1
Enter the size of set 1
3
Enter the element of set 1
1
2
3
Enter the size of set 2
2
Enter the element of set 2
2
3
Union : 1 2 3

Press 1 for union
Press 2 for Intersection
Press 3 for Subtraction
Press 4 for Exit

Enter choice 2
Enter size of set 1
3
Enter the element of set 1
1
2
3
```

Enter the size of set 2

2

Enter the element of set 2

3

4

Intersection : 3

Press 1 for union

Press 2 for intersection

press 3 for subtraction

Press 4 for exit

Enter your choice

Enter the size of set 1

3

Enter the element of set 1

1

2

3

Enter the size of set 2

2

Enter the element of set 2

8

2

difference : 1

Press 1 for union

Press 2 for intersection

press 3 for subtraction

Press 4 for exit

Enter choice : 4

Program 6

Aim: To perform Binary search tree operations

Algorithm:

Step 1 : Start

Step 2 : Declare a structure and structure pointers for insertion, deletion and search operations and also declare a function for inorder traversal.

Step 3 : Declare a pointer as root and also the required variable.

Step 4 : Read the choice from the user to perform insertion, deletion, searching and inorder traversal.

Step 5 : If the user choose to perform insertion operation then read the value which is to be inserted to the tree from the user.

Step 5.1 : pass the value to the insert pointer and also the root pointer.

Step 5.2 : check if root then allocate memory for the root

Step 5.3 : Set the value to the info part of the root and then set left and right part of the root to null and return root.

Step 5.4 : check if root → info > x then call the insert pointer to insert to left of the root.

Step 5.5 : check if root → info > x then call the

insert pointer to insert to the right of the root.

Step 5.6 : Return the root.

Step 6 : If the user choose to perform deletion operation then read the element to be deleted from the tree the root pointer and the item to the delete pointer

Step 6.1 : check if not ptr then print node not found.

Step 6.2 : check if not ptr then print no else if ptr → info < x then call delete pointer by passing the right pointer and the item.

Step 6.3 : Else if ptr → info > x then call delete pointer by passing the left pointer and the item.

Step 6.4 : check if ptr → info == item then check if ptr → left == ptr → right then free ptr and return null.

Step 6.5 : Else if ptr → left == null then set P_1. ptr → right and free ptr, return P_1.

Step 6.6 : Else if ptr → right == null the set P_1 = ptr → left and free ptr, return P_1

Step 6.7 : Else Set $P_1 = Ptr \rightarrow right$ and
$P_2 = Ptr \rightarrow right$.

Step 6.8 : while $P_1 \rightarrow left$ not equal to null,
Set P_1 $left$ $P_1 \rightarrow left$ and free Ptr,
return P_2.

Step 6.9 : return Ptr.

Step 7 : If the user choose to perform Search operation the call the pointer to perform search operation.

Step 7.1 : Declare the necessary pointer and variables.

Step 7.2 : Read the element to be searched

Step 7.3 : while Ptr check if Item > $Ptr \rightarrow$ info then $Ptr = Ptr \rightarrow right$.

Step 7.4 : else if Item < $Ptr \rightarrow$ info then
$Ptr = Ptr \rightarrow left$.

Step 7.5 : Else break.

Step 7.6 : check if Ptr then Print that the element is found.

Step 7.7 : Else Print element not found in tree and return root.

Step 8 : If the user choose to perform traversal then call the traversal funtion and pass the root pointers.

Step 3.1 : if root not equal to null recursively call the function by passing root → left.

Step 3.2 : print root → info

Step 3.3 : Call the traversal function recursively by passing root → right.

Step 4 : End

output

1. insert in Binary tree
2. Delete from binary tree
3. inorder traversal of Binary tree
4. Search
5. Exit

Enter choice : 1

Enter new element : 20

Root is 20

inorder traversal of binary tree is : 20

1. Insert in Binary tree
2. Delete from Binary tree
3. inorder traversal of Binary tree
4. Search
5. Exit

Enter choice : 1

Enter new element : 25

inorder traversal of binary tree is : 20 25

1. Insert in Binary tree
2. Delete from Binary tree
3. Inorder Traversal of Binary tree
4. Search
5. Exit

Enter choice : 4

Search operation in binary tree

Enter the element to be searched : 25

Element 85 which was searched is found.

1. Insert in Binary tree
2. Delete from Binary tree
3. Inorder traversal of Binary tree
4. Search
5. Exit

Enter choice : 5

End of the program

Program : 7

Aim : To perform Disjoint set operations.

Algorithm

Step 1 : Start

Step 2 : Declare the structure and related variable.

Step 3 : Declare function makeset()

Step 3.1 : Repeat step 3.2 to 3.4 until i<n.

Step 3.2 : disparent [i] is set to i.

Step 3.3 : set disrank [i] is equal to 0.

Step 3.4 : increment by 1.

Step 4 : Declare a function display set.

Step 4.1 : Repeat step 4.2 to 4.3 until i<n

Step 4.2 : print disparent [i]

Step 4.3 : increment i by 1.

Step 4.4 : Repeat step 4.5 and 4.6 until i<n

Step 4.5 : print disrank [i]

Step 4.6 : increment i by 1.

Step 5 : Declare a function find and pass x to the function.

Step 5.1 : check if disparent [n] = x then set the return value to disparent [x]

Step 5.2 : return disparent [x]

Step 6 : Declare a function union and pass to variables x and y.

Step 6.1 : Set x set to Find (x).

Step 6.2 : Set y set to Find (y).

Step 6.3 : check if y set == x set then return

Step 6.4 : check if disrank [x set] < disrank [y set] then

Step 6.5 : Set y set = disparent [y set]

Step 6.6 : Set -1 to disrank [x set]

Step 6.7 : Else if check disrank [x set] > disparent [y set]

Step 6.8 : Set x set to disrank [y set]

Step 6.9 : Set -1 to disrank [y set]

Step 6.10 : Else disparent [y set] = x set.

Step 6.11 : Set disparent [x set]+1 to disrank [x set]

Step 6.12 : Set 1 to disrank [y set]

Step 7 : Read the number of elements.

Step 8 : Call the function makeset.

Step 9 : Read the choice from user to perform union find and display operation.

Step 10 : If the user choose to perform union operation read the element to perform union and then call the function to perform union operation.

Step 11 : If the user choose to perform Find operation read the element to check if connected.

Step 11.1 : Check if find (x) == Find(y) then print connected component.

Step 11.2 : Else print not connected component.

Step 12 : If the user choose to perform display operation then call the display set function.

Step 13 : End.

Output

How many elements : 4

Menu

1. union

2. Find

3. Display

Enter choice : 1

Enter elements to perform union:

3

4

Do you wish to continue : (Y/N)

Y

Menu

1 union

2 Find

3 Display

Enter choice : 1

Enter element to perform union:

5

6

Do you wish to continue Y/N)

Y

Menu

1. Union
2. Read
3. Display

Enter choice : 3

Parent Array

2 1 2 3

Rank Array

-1 0 0 1

Do you wish to continue ? (Y/n)

n