AIM:- Merge two sorted arrays and store in a third array.

Step 1:- Start.

Step 2:- Declare the variable.

Step 3:- Read the size of the first array.

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

Step 5:- Read the size of the second array

Step 6:- Read the elements of second array in sorted array 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[i++]

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:- End.

Output:-

Enter number of elements in array 1: 4

Enter array 1 in sorted order: 1 3 5 7

Enter number of elements in array 2: 4.

Enter array 2 in sorted order: 1 4 6 7

first array is: 1 3 5 7.

Second array is: 2 4 6 7

Merged array is: 1 2 3 4 5 6 7 7.

AIM:- circular queue - Add, Delete, Search.

Step 1:- Start.

step 2:- Declare the queue and other variables.

Step 8:- 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 && rear == -1 then set front=0, rear=0 and set queue [rear] = element.

Step 8.2:- else if rear + 1 % max == front or front == rear + 1 then print queue is overflow.

Step 5.8:- Else set rear = rear + 1 % max & set queue [rear] = element.

Step 6:- If the user choice is the option dequeue 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 elemented then set front = -1 and rear = -1

Step 6.3:- else print the element to be dequeue used 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+1 % max.

Step 8:- if the user choose the search.

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 it position and increment by 1.

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

Step 9: End.

Output:-

choices:-

1. Insert
2. Delede.
3. Display.
4. Search.
5. Exit

Enter your choice:-

1

Enter the number to be inserted.

7.

choices:-

1. Insert
2. Delede.
3. Display.
4. Search.
5. Exit.

Enter your choice:

1.

Enter the number to be inserted:

14.

choises:-
1. insert
2. Delete
3. Display
4. Search
5. Exit
Enter your choice: 2.
4 was deleted.

choises:-
1. insert
2. Delete
3. Display
4. Search
5. exit
Enter your choice: 3.

14
choises:-
1. insert
2. Delete
3. Display
4. Search
5. exit
Enter your choice:-
4.
Enter the element to be search: 14
Item found at location 2.

AIM:-

Singly linked stack.

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 for the user.

Step 5:- If the user choose to push an element, then read the element to be push & call the functions to push, the element by passing the values to the function.

Step 5.1:- Declare the newnode & allocate memory for the newnode.

Step 5.2:- Set newNode → data = value.

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

Step 5.4:- Set newnode → next = top.

Step 5.5:- Set top = newNode & then print. insertion is successful.

Step 6:- If the 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 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 6.5:- free the temp
step 7:- if the user choose the display then call the function to display the element in 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 step below while temp → next != null
step 7.4:- Print temp → data.
step 7.5:- set 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 variables.
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 ptr1 != null.

Step 8.6:- check if ptr1 → data == item then print element founded and to be located and set flag = 1.

Step 8.7:- Else set flag = 0.

Step 8.8:- Increment i by 1 and set ptr1 = ptr1 → next.

Step 8.9:- check if flag = 0 then print the element not found.

Step 9:- End.

output:-

choices:-
1. push.
2. pop
3. Display
4. Seazch
5. Exit.
Enter your choice: 1.
Enter the value to be Pnseat: 7.
Pnseation is Sauensfull.

choices:
1. push.
2. pop
3. Display.
4. Search.
5. Exit.
Enter your choice: 1.
Enter the value to be Pnseat: 14.
Pnseation is Successfull.

choices:
1. Push.
2. POP.
3. Display.
4. Search.
5. Exit.
Enter your choice: 2.
popped element: 14.

choices:-

1. push.
2. POP.
3. Display.
4. Search.
5. Exit.

Enter your choice: 3.

7 --→ Null

choices:-

1. push.
2. POP.
3. Display.
4. Search.
5. Exit.

Enter your choice: 4.

Enter the item to be searched: 2.

item not found.

choices:-

1. push.
2. POP.
3. Display.
4. Search.
5. Exit.

Enter your choice: 5.

AIM:- Doubly linked Stack:-

Step 1: Start.

Step 2: Declare a structure and related variables.

Step 3: Declare functions to create a node insert a node in the beginning 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 values to be inserted 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 create a node, perform step 4 to 4.3.

Step 6.2 :- Set head = temp and temp1 = head

Step 6.3 :- else call the function to create a node, perform step 4 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 end.

Step 7.1 :- check if head == null then call the functions to create a newnode 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. then → 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 variable.

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

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

Step 8.4 :- check if head == null and pos = 1 then print "Empty list" can't insert other than 1st position.

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

Step 8.6 :- While i < pos then set temp2 = temp2 → next then increment i by 1.

Step 8.7 :- call the function to create a newnode 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 then 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 deleted set temp2 = head.

Step 9.3:- check if pos < 1 or pos >= count + 1, then print position out of range.

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

Step 9.5:- while i < pos then temp2 = temp2 → next. and increament i by 1.

Step 9.6:- check if i == 1 then check if temp2 → next == null then point node delete free (temp2). Set temp 2 = 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 → next → prev = temp 2 → prev then check if i = 1 then temp2 → prev → next = temp2 → next.

Step 9.9:- check if i == 1 then head = temp2 → 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 temp.a = u.

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

Step 10.3:- while temp2→next1 = null then
printf temp2→n then temp2 =
temp2→next.

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

Step 11.1:- Declare the necessary variables.

Step 11.2:- Set temp2 = head.

Step 11.3:- check if temp2 == null then printf the list is empty.

Step 11:4:- Read the value to be searched.

Step 11.5:- while temp2 != null the check if
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 found in the
list.

12: End.

Output:-

1. Insert at beginning.
2. Insert at End.
3. Insert at position 1.
4. Delete at i
5. Display from beginning.
6. Search for element
7. Exit

Enter choice: 1.
Enter value to node: 3.
Enter choice: 1
Enter value to node: 4.
Enter choice: 5.
Linked list element from beginning: 4 3.
Enter choice: 3.
Enter the position to be inserted: 2.
Enter value to node: 1.
Enter choice: 2.
Enter value to node: 6.
Enter choice: 5.
linked list elements from beginning:
4 1 3 6.
Enter choice: 4.
Enter position to be deleted: 1.
Node deleted.

Enter choice:5.
Linked list elements from biginning: 1 3 6
Enter choice : 4.
Enter position to be deleted: 3.
Node deleded from the list.
enter choice : 5.
linked list element from begining : 1, 3
Enter choice: 6.
Enter value to be search : 3.
Data found in 2 position.
Enter choice:6.
Enter value to be search: 7.
Error 7 not found in list.
Enter choice : 7.

AIM:-

Binary search tree:-

Step 1: Start

Step 2: Declare a structure and structure pointers for insertion

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 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 > t 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 righ 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 pass the root pointer and the item to the delete pointer.

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

Step 6.2 :- 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 ~~right~~ 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 p1 ptr → right and free ptr; return p1

Step 6.6 :- else if ptr → right == null then set p1 = ptr → left and free ptr return p1

Step 6.7 :- else set p1 = ptr → right and p2 = ptr → right.

Step 6.8:- while p1 → left not equal do null,
set p1 → left. ptr → left and free ptr,
return p2.
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 → info.
then ptr = ptr → right.
Step 7.4:- ~~while~~ else if item < ptr → info
then ptr = ptr → left.
Step 7.5:- else break.
Step 7.6:- check if ptr then print that the element is not 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 function and pass the root pointer.
Step 8.1:- If root not equal do null recursively call the function by passing root → left.
Step 8.2:- Print root → info.
Step 8.3:- call the traversal function recursively by passing root → right.

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: 10.

root is 10.

Inorder traversal of binary tree is: 10.

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: 12.

root is 10.

Inorder traversal of binary tree: 10 12.

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: 16.
Root is 10.
Inorder traversal of binary tree: 10 12 16.
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: 14.
Inorder traversal of binary tree: 10 12 14 16.
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 : 15.
Root is 10.
Inorder traversal of binary tree: 10 12 14 15 16
1. Insert in Binary tree.
2. Delete from Binary tree.
3. Inorder traversal of binary tree.
4. Search.
5. Exit.
Enter choice : 2.

Enter the element to be deleted: 15 to
10 12 14 16.
1. Insert in binary Tree.
2. Delete Binary tree.
3. Inorder traversal of binary tree.
4. Search.
5. Exit
Enter choice: 3.
Inorder traversal of binary tree is:-
10 12 14 16.
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: 16.
Element 16 which was searched is found and
15-16.
1. Insert in binary tree.
2. Delete from binary tree.
3. Inorder traversal of binary tree.
4. Search.
5. Exit.
Enter choice: 5.

AIM:- Set operations:-

Step1:- Start.
Step2:- Declare the necessary variable.
Step3:- Read the choice from the user to perform set operation.
Step4:- If the user choose to perform union.
Step 4.1:- Read the cardinality of 2 sets.
Step 4.2:- check if m1=n then print can't perform union.
Step 4.3:- 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|=n then print can't perform Intersection.
Step 5.3:- Else read the elements 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 operations.

Step 6.1:- Read the cardinality of a set.
Step 6.2: check if m!=n then print can't 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 then C[i]=0.
Step 6.7: else C[i]=1.
Step 6.8:- Increment i by 1.
Step 7: Repeat the step 7.1 and 7.2 until i<m
Step 7.1: print C[i].
Step 7.2: increment i by 1.

Output:-

```
Choice do perform.

1. union.
2. intersection
3. Difference
4. Exit.
Choice: 1.
Enter first set: 3.
Enter second set: 3.
Enter first set(0/1): 1.
0
1
Enter set:(0/1) 0 0 1
Elements of set 1 union set 2: 1 0 1.
Choice do perform:
1. Union 2. intersection. 3. Difference 4. Exit.
Choice : 3.
Enter first set: 4.
Enter second set: 4.
Enter first set: (0/1) 1
0
0
1
Enter second set:(0/1) 1.
0
1
0
Elements of set 1 - set 2: 0 0 0 1
```

choice to peaform:-
1. union 2. intersection. 3. Difference 4. Exit.
choice: 2.
Enter first set: 3.
Enter second set: 3.
Enter first set: (0/1) 1
0
0

Enter second se: (0/1)
1
0
1
Elements of set 1
intersection set 2 1 0 0.
choice to peaform:-
1. union 2-intersection. 3. Difference. 4. Exit.
choice: 4.
Program exit successfully.

AIM:-

Disjoin Set and the associated operations (~~insertion~~ Create, Union, find).

Step 1:- Start.

Step 2: Read the number of elements from the user and Store ~~and~~ it to the dis n.

Step 3: call function make set() then.

Step4: Set i=0.

Step5: Repeate for i< dis.n then.

Set dis.parent[c] = i

Set dis rank[i] = 0

Set i=i+1

[End for loop].

Step6: User selects the union operation then.

Step7: Read the elements to perform union and Store into x and y respectively.

Step 8: perform find operation in with x and y Store result into x set and y set perform step 2 3.

Step 9: if x set == y set then.

[end y]

Step 10: if dis rank[x set] < dis.rank[y set]

then: set dis.parent[x set] = y set.

set dis rank[x set] = -1.

[end if].

Step 11: Else if dis.rank [x set] > dis.rank [y set]
then.
SET dis.parent [y set] = x set;
SET dis.rank [y set] = -1;
[end of 16]
Step 12: Else.
Step 13: SET dis parent [y set] = x set
SET dis rank [x set] = dis rank [x set] + 1
SET dis rank [y set] = -1.
Step 14: If user choose find operation then.
Step 15: Read the elements to check it and store the value into the variables x and y respectively.
Step 16: If find x == find y then.
Display "connected components".
Step 17: else
Display "Not connected components".
Step 18: If user select the display operation then.
Step 19: set i = 0.
Step 20. Repeade for i < dis n then.
print dis.parent [i]
set i = i + 1.
[end of for loop].
Step 21: If dis.parent [x] != x
then:
set dis.parent [x] = find (dis par [x])
return dis.parent [x].
Step 22: Exit.

Output:-

```
How many elements : 5.
-- Menu --
1. union.
2. find.
3. Display.
Enter choice.
 1.
Enter the elements to perform union = 3
Do you wish to continue ? (Y/O) : 1
-- menu --
1. Union.
2. find.
3. Display.
Enter choice.
 3.
parent array.
0 1 2 2 4.
rank array
0 0 1 -1 0.
Do you want do continue ? (Y/O)
 1
-- menu --
1. Union.
2. find.
3. Display.
```

Enter choice :
2
Enter the elements to check if connected
components.
2
3.
connected components.
Do you wish to continue? (1/0)
1
— — menu — —
1. Union.
2. find.
3. Display.
Enter choice.
2.
Enter the elements to check it connected
components 1.
3.
connected components.
Do you wish to continue? (1/0)
0.