Program NO: 1

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

Step 1: Start

Step 2: Declare the variables.

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 elements of second array in Second order.

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

Step 8: check if array1[i] < array2[j] then

res[k] = array1[i]

i++

k++

Step 9 : else res[k] = array2[j]

j++

k++

Step 10 : Repeat step 11 while i<m

Step 11 : res[k] = array1[i]

i++

k++

Step 12 : Repeat step 13 while j<n

Step 13 : res[k] = array2[j]

j++

k++

Step 14 : Print the merged array

Step 15 : End.

Output

```
Enter the number of element in the array1
3

Enter array1 in sorted order
1
5
9

Enter the number of element in the array2
2

Enter array2 in sorted order
3
6

Merged Array is
1 3 5 6 9
```

Program NO: 2.

AIM: Singly linked stack - Push, Pop, linear search.

Step 1: Start

Step 2: Declare the node and the variable.

Step 3: Display the functions for push, pop, display and search the elements.

Step 4: Read the choice from the user.

Step 5: If the user choose to push a 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 newNode and allocate Memory.

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 and then print insertion is successful.

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 option then call the function to display the element in the stack.

Step 7.1 : check if top == Null then print the stack is empty.

Stack 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 : set temp = temp → next.

Step 8 : if the user choose to search option then call the function to search an element.

Step 8.1 : Declare a pointer variable ptr and variables.

Step 8.2 : Initialize ptr = top.

Step 8.3 : check if ptr = null then print list is 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 the element found at the location and set flag = 1

Step 8.7 : Else set flag = 0

Step 8.8 : Increment the i value by 1 and set ptr = ptr->next.

Step 9 : End.

Output

Enter your choice

1. Push
2. POP
3. display
4. Seach
5. Exit

Enter you choice: 1

Enter the value to be inserted: 2

Insertion is success!!

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

Enter you choice: 1

Enter the value to be inserted: 3

Insertion is success!!

1. Push
2. POP
3. display
4. seach
5. Exit

Enter your choice : 3

3 ---> 2 ---> NULL

1. Push
2. POP
3. display
4. seach
5. Exit

Enter your choice : 4.

Enter item which you want to seach : 3

Item found at location 1.

1. Push
2. POP
3. Display
4. search
5. Exit

Enter you choice: 2

Deleted element: 3

Program NO: 3

AIM : Circular queue operation - Add, delete, search.

Step 1 : Start

Step 2 : Declare the queue and the 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 inseted from the user and call the enqueue function by passing value.

Step 5.1 : check if front == 0 && rear = max-1 || Front == rear + 1

Print the circular queue is full.

Step 5.2 : Else if Front = -1 then set Front = Front + 1

Step 5.3 : check if rear == max-1 and set rear = 0

Step 5.4 : Else rear = rear + 1 and set
queue[rear] = m

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

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

Step 6.2 : Else check if Front = Max - 1 and set Front = 0

Step 6.3 : else if (Front == rear) and set front = -1 and rear = -1

Step 6.4 : else front = Front + 1
Print the element was deleted.

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

Step 7.1 : check if front = -1, then print the queue is empty.

Step 7.2 : Repeat the step 7.3 while P <= rear

Step 7.3 : Print queue[P]

Step 7.4 : else while P<max-1 then print queue[P++] set P=0

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 : Repeat step 8.3 then For(P=front; i<=rear; i++)

Step 8.3 : check if element = queue[P] then print the element present at the location and increment c by 1

Step 8.4 : check if c=0 then print the element not found.

Step 9 : End.

Output

Menu

1. Insert
2. Delete
3. Display
4. Search

Select : 1

Enter the number to be inserted: 10

Menu

1. Insert
2. Delete
3. Display
4. Search

Select : 3

10

Menu

1. Insert
2. Delete
3. Display
4. Search.

Select : 4

Enter the element to be searched : 10

The element is present at the location 1

Menu :

1. Insert
2. Delete
3. Display
4. Search

Select : 2

10 was deleted.

Program NO: 4

AIM: Doubly linked list operation - insertion, deletion, search.

Step 1: Start

Step 2: Declare a structure and variables.

Step 3: Declare functions to create a node, insert a node at 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 variable.

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 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 the end.

Step 7.1 : check if head==null then call the function to create a newNode then set temp=head and then set head=temp1.

Step 7.2 : Else call the function to create a newnode then set temp1->next=temp temp->prev=temp1 and temp1=temp.

Step 8 : IF the usea choose to peaform insertion in the list at any position then call the function to peaform the inseation operation.

Step 8.1 : Declare vaiable.

Step 8.2 : Read the position whaie the node need to the inseated, set temp2=head.

Step 8.3 : check if pos<1 oa pos>count+1 then paint the position is out of aange.

Step 8.4 : check if head==null and pos=1 then call the function to ceate newNode then set temp=head and head=temp1.

Step 8.5: while i<Pos then set temp2 = temp2 →next then increment i by 1.

Step 8.6: Call the function to create a new node and then set temp→Prev = temp2

temp→next = temp2→next

temp2→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 variable.

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 point 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 increment i by 1.

Step 9.6 : check if i==1 then check if
tempa2->next ==null then print node
deleted free(tempa2) set tempa2=head=null

Step 9.7 : check if tempa2->next ==Null then
tempa2->Prev->next=null then
Free(tempa2. Then print node deleted.

step 9.8 : tempa2->next->prev = tempa2->Prev
then check if i==1 then tempa2->
prev->next = tempa2->next.

step 9.9 : check if i==1 then head=tempa2->next
then print node deleted then free(tempa2)
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 tempa2 =n

step 10.2 : check if tempa2=null then print
list is empty.

Step 10.3 : while tempa->next = null then point tempa->n then tempa = tempa->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 variable.

Step 11.2 : Set tempa=>head

Step 11.3 : check if tempa == null then print the list is empty.

Step 11.4 : Read the value to be searched

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

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

Step 11.7 : Print element not found in the list

Step 12 : End.

Output

1. Inserted at beginning
2. Insert at end
3. Insert at specific position
4. Delete at specific position
5. Display from beginning
6. Search for element
7. Exit.

Enter you choice: 1

Enter the value to node: 10

Enter you choice: 1

Enter the value to node: 20

Enter you choice: 2.

Enter value to node: 30

Enter you choice: 3

Enter position to be inserted: 2

Enter value to node: 40

Enter your choice: 5

Linked list element from beginning:

20 40 10 30

Enter your choice: 4

Enter the position to be deleted: 3

node deleted

Enter your choice: 5

Linked list element from beginning:

20 40 30

Program NO: 5.

AIM: Set data structure and set operations using bit string.

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 union operation.

Step 4.1: Read the cardinality of 2 sets.

Step 4.2: check if m!=n then print cannot perform union.

Step 4.3: Else read the element in both the set

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 cannot perform intersection.

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

Step 5.4 : Repeat 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 2 sets.

Step 6.2 : check if m!=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 then 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

```
Menu:
1. union
2. Intersection
3. Difference
4. Exit
choice: 1
Enter the size of first set: 2
Enter the size of second set: 2
Enter the elements of first set:
1 2
Enter the elements of second set:
3 4
Elements of set1 union set2 : 3 6
Menu
1. Union
2. Intersection
3. Difference
4. Exit.
```

Choice:2:

Enter size of first set: 2

Enter size of second set: 2

Enter the element of the first set:

1 2

Enter the element of the second set:

2 3

Enter the element of set1 intersection set2: 0 2

Menu

1. Union
2. Intersection
3. Difference
4. Exit

Choice: 3

Enter the size of first set: 2

Enter the size of second set: 2

Enter the element of the first set:

1 3

Enter the element of the second set: 2 3

Elements of set1 - set2 : 1 1

Program NO: 6

AIM: Binary search tree - insertion, deletion, search.

step 1: Start.

step 2: Declare a structure and pointers for insertion deletion and search operation 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 user.

step 5.1: Give the value to be inserted 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 part of the root and then set left and right part of the root to null and return root.

Step 5.4 : check if *node == null set *node = createNode (data)

Step 5.5 : Else if data < *node → data Insert at a the left of the root.

Step 5.6 : Else if data > *node → data, Insert at the right of the root

Step 5.6 : Return root.

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

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

Step 6.2: Else if ptr→data < data then
call delete pointer by passing
the right pointer and the item.

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

Step 6.4: check if ptr→data == 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→ ringht and free ptr
and return p_1.

Step 6.6: Else if ptr→right == null then
set p_1 = ptr→left and free ptr
return p_1.

Step 6.7: Else set p_1 = ptr→right and
p_2 = ptr→right

Step 6.8: while p_1→left not equal to null
set p_1→left, ptr→left and free
ptr return p_2.

Step 6.9 : return Ptr.

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

Step 7.1 : Check if t= node

Step 7.2 : else if data < node->data
FindElement(node->left ;data)

Step 7.3 : Else if data data > node->data
FindElement(node->right, data)

Step 7.4 : Print the data found, node->data.

Step 8 : If the user choose to perform traversal then call traversal Function and pass pointer variable.

Step 8.1 : If node != Null recursively call the function by passing node->left.

Step 8.2 : Print node->data

Step 8.3 : call the traversal function recursively by passing node->right

Step 9 : Stop.

Output

1. Insertion in BST
2. Deletion in BST
3. Search an element in BST
4. Inorder traversal
5. exit

Enter your choice: 1

Enter your data : 10

Continue insertion (0/1): 1

Enter your data : 20

Continue insertion (0/1): 1

Enter your data : 30

Continue insertion (0/1): 0

1. Insertion in BST
2. Deletion in BST
3. search an element in BST
4. Inorder traversal
5. Exit

Enter your choice: 4

inorder traversal:

10 20 30

1. Insertion in BST
2. Deletion in BST
3. Search Element in BST
4. Inorder Traversal
5. Exit.

Enter your choice: 3.

Enter the value for data: 10

Data found

1. Insertion in BST
2. Deletion in BST
3. Search Element in BST
4. Inorder traversal
5. Exit

Enter your choice: 2

Enter your data: 20

Program NO: 7

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

Step 1: Start.

Step 2: Declare a structure ~~structure~~ and declare the sets

Step 3: Read the choice and perform union, find and display functions.

Step 4: If first create a n single item sets.

Step 4.1: for (int i=0; i<dis.n; i++)
set dis.Parent[i] = i;
dis.rank[i]=0

Step 5: If the user select the display option then perform the display function.

Step 5.1: Print the parent array.

Step 5.2: for (int i=0; i<dis.n; i++)
Print dis.parent[i]

Step 5.3: Print the rank of the array

Step 5.4: for(int i=0; i<dis.n ; i++)

Print dis.rank[i]

Step 6 : If the user choose the Find option

Step 6.1 : check if dis.parent[x] != x

set dis.Parent[x] = Find(dis.parent[x])

Step 6.2: Return dis.Parent[x]

Step 7 : If the user choose the union option,

Step 7.1 : declare the sets

Step 7.2: check if(xset==yset)

Step 7.3: check if dis.rank[xset] < dis.rank[yset]

set dis.parent[xset] = yset

dis.rank[xset] = -1

Step 7.4 : else if dis.rank[xset] > dis.rank [yset]

set dis.Parent[yset] = xset

dis.rank[yset] = -1.

Step 7.5 : else set

dis.Parent[Yset] = Xset

dis.rank[Xset] = dis.rank[Yset] + 1

dis.rank[Yset] = -1

Step 8 : Stop.

Output

Enter the number of elements: 5

1. union
2. Find
3. display

Enter your choice: 3

Parent array

0 1 2 3 4

Rank of array

0 0 0 0 0

Do you want to continue(0/1): 1

1. union
2. Find
3. display

Enter your choice: 1

Enter the element to perform union:

0 1

Do you want to continue (0/1): 1

1. union
2. Find
3. display
Enter you choice: 3

0 0 2 3 4.