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 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[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: 2 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 3: Declare the junctions for enqueue, de-que, 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 q; 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 & 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 deleted
then set front = -1 and rear = -1

step 6.3: Else print the element to be deque-
ued set front = front + 1 % max

step 7: If the user choice is to display the
Queue then call the junction 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 then
call the junction to search an element
in the queue.

step 8.1: Read the element to be sear-
ched 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: End

Output

```
Choices:
1. Insext
2. Delete
3. Display
4. Seauch
5. Exit
Enter your choice:
1
Enter the number to be inserted:
7
choices:
1. Insext
2. Delete
3. Display
4. Seauch
5. Exit
Enter your choice:
1
Enter the number to be inserted:
14
choices:
1. Insext
2. Delete
3. Display
4. Seauch
5. Exit
Enter your choice:
2
```

7 was deleted.
choices:
1. Insext
2. Delete
3 Display
4. Seauch
5. Exit
Enter your choice :
3
14
choices:
1. Insext
2. Delete
3. Display
4. Seaxch
5- Exit
Enter your choice :
4
Enter the element to be searched: 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 from the user

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

step 5.1 : Declare the newnode & allocate memory for the newnode

step 5.2 : Set newNode → data = value.

step 5.3 : ~~set~~ 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 6.4: 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~~ 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 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 8.7 : Else set flag=0.

step 8.8 : Increment i by 1 and set ptr=ptr→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. Search
5. Exit
Enter your choice: 1
Enter the value to be insext: 7
Insertion is successfull
choices:
1. Push
2. Pop
3. Display
4. Seauch
5. Exit
Enter you choice: 1
Enter the value to be insert: 14
Insertion is successfull.
choices:
1. Push
2. Pop
3. Display
4. Seaxch
5. Exit
Enlex you choice: 2
Popped element: 14

Choices:

1. ~~Insert~~ Push
2. Pop
3. Display
4. Seauch
5. Exit

Enter your choice: 3

7 -> NULL

choices:

1. Push
2. Pop
3. Display
4. Seauch
5. Exit

Enter your choice: 4

Enter the item to be searched: 2

item not found.

choices:

1. Push
2. Pop
3. Display
4. Seauch
5. Exit

Enter your choice: 5

AIM: Doubly Linked list

Step 1 : start

step 2 : Declare a structure and related Vaiables

step 3 : Declare junctions 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 junction 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 inserted to the node.

step 4.3 : set temp → n = data and increement 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 junction 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 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 variable.

Step 8.2: Read the position where the node need to be 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: Check if head==null and pos=1 then print "Empty list" cannot insert other than 1st position".

step 8.5: Check if head==null and pos=1 then call the junction to create newNode, 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 junction to create a newonode 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 junction 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 ~~temp~~ i<pos then temp2=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->next->prev=temp2->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 temp2=n

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

Step 10.3 : while temp2->next1 = 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 operation.

Step 11.1 : Declare the necessary variables

Step 11.2 : set temp2=head

Step 11.3 : Check if temp2==null then print the list is empty.

Step 11.4 : Read the value to be searched.

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

Step 12 : End.

Output

1. Insert at beginning
2. Insert at end
3. Insert at position
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 elements from beginning: 4 3
Enter choice: 2
Enter value to node: 6
Enter choice: 5
Linked list element from beginning: 4 3 6
Enter choice: 3
Enter position to be inserted: 2
Enter value to node: 1
Enter choice: 5
Linked list element From beginning: 4 1 3 6
Enter choice: 4
Enter position to be deleted: 1

Node deleted
Enter choice : 5
Linked list elements from beginning : 1 3 6
Enter choice : 4
Enter position to be deleted: 3
Node deleted from list
Enter choice : 5
Linked list elements from beginning: 1 3
Enter choice : 6
Enter value to search: 3
Data found in 2 position
Enter value to search : 7
Error : 7 not found in list
Enter choice : 7

Pgm No: 3

AIM: Binary Search Trees.

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 ~~self~~ set left and right part of the root to null and return root.

step 5.4: Check if root → info > e 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: Retuen 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 not 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 left pointer and the item.

step 6.4: Check if ptr→info ==item then check if pte→left==pte→right then tree ptr and retuen null.

step 6.5: Else if pte→left==null then set p1.ptr→right and free ptr; retuen p1

step 6.6: Else if ptr→right==null then set p1=pte→left and free pte, retuen p1.

step 6.7 : Else set p1 = ptr→ right and p2 = ptr→ right

step 6.8 : while p1→ left not equal to null, set p1 → left . ptr→ left and free ptr, return p2

step 6.9 : Return pte.

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

step 7.1 : Declare the ~~mem~~ necessary pointers and variables

step 7.2 : Read the element to be ~~sh~~ searched

step 7.3 : while ptr check if item > ptr→ info then ptr = ptr→ right.

step 7.4 : Else if item < pte→ info then pte = pte→ left.

step 7.5 : Else break

step 7.6 : Check if pte 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 function and pass the root pointers.

step 8.1: if root not equals to null recur.
sively call the functions by passing
root→lgt

step 8.2: print root→info

step 8.3: call the traversal function recursiv.
ely 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
root is 10
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 :- 45

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 : 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 is - 16.

1. Insert in Binary Tree
2. Delete From Binary Tree
3. Inorder traversal of Binary Tree
4. Search
5. Exit

Enter choice : 5

Pgm No: 6

AIM : Set operations

Steep 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!=n then print cannot 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 cannot perform intersection.

Step 5.3: Else read the elements is 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 : Inurement i by 1.

step 6 : If the user choose to perform set difference operation.

step 6.1 : Read the cardinality of 2 set

step 6.2 : Check if m!=n then print cannot peform 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 to perform:
1. Union 2. Intersection 3. Difference 4. Exit
choice: 1
Enter first set: 3
Enter first second set: 3
Enter first set:(0/1) 1
0
1
Enter set:(0/1) 0 0 1
Elements of set1 union set2 : 1 0 1
choice to 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
0
Elements of set1-set2 : 0 0 0 1
choice to perform:
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 set : (0/0) 1 0 1

Elements of set 1 intersection set 2: 1 0 0

choice to perform:

1. Union 2. Intersection 3 Difference 4. Exit

choice: 4

Program exit successfully!

Pgm No : 7

AIM : Disjoint set and the associated opera-tions(create, union, find)

Step 1 : start

step 2 : Read the no of elements from the user and store it in to the dis.n

step 3 : call function makeset() then

set 4 : set i=0

step 5 : Repeat for i < dis.n then

set dis.parent [i] = i

set dis.rank [i] = 0

set i = i+1

[end for loop]

step 6 : User selects the union operation then:

step 7 : Read the elements to perform union and store into x and y respectively.

step 8 : // perform find operation ib with x and y. store result into x set and y set perform step 23

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 [xset] = -1

[end if]

step 11 : Else if dis.rank [xset] > dis.rank [yset] then:

SET dis.parent [yset] = xset;

SET dis.rank [yset] = -1;

[End of IF]

Step 12 : Else

step 13 : SET dis.parent [yset] = xset

SET dis rank [xset] = dis.rank [xset] + 1

SET dis.rank [yset] = -1

Step 14 : If user choose find operation then:

Step 15 : Read the elements to check H 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 : Repeat for i < dis.n then

Print dis.parent [i]

Set i = i + 1

[End of for loop]

Step 23: if dis.parent[x] != x
then:
SET dis.parent[x] = find(dis.parent[x])
return dis.parent[x]

Step 24: Exit.

Output

How many elements? 5

--menu--

1 Union

2. Find

3. Display

Enter choice

1

Enter the elements to perform union:2

3

Do you wish to continue? (1/10)

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 wish to continue? (1/10)

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? (1/10)
1

--menu--
1-union
2. Find
3. Display
Enter choice
2
Enter the elements to check if connected components
1
3
connected components
Do you wish to continue? (1/10)
0