Qn1. Merge two sorted arrays

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 ~~elements of second array in sorted order.~~

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

step 7: Repeat step 8 & 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: Stop.

Output

```
size of First array: 2
Enter value in sorted order:
3
6
size of second array : 4
Enter value in sorted order:
4
5
7
8

First Array:
4 5 7 8 3 6
Second Array:
4 5 7 8
Merged Array:
3 4 5 6 7 8
```

2. Stack operations

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 the read the element to be pushed & call the function to push the element by passing value to the function.

step 5.1 : Declare the new node & allocate memory for the new node

step 5.2 : Set new node -> data = value

step 5.3 : check if top == null then set new node -> next = null

step 5.4 : Set new node -> next = top

step 5.5 : Set top = new node & 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 6.0~~:

step 6.1: check if top == Null, then print stack is empty

step 6.2: else declare a pointer variable temp and initialise 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 display function.

step 7.1: check if top == Null, then print stack is empty

step 7.2: ~~the~~ else declare a pointer variable temp & initialize it to top

step 7.3: Repeat steps below while temp -> next != nu

step 7.4: print temp -> data

step 7.5: set temp = temp -> next

step 8: If the user choose to search an element, call the search function

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 ~~until~~ while ptr != null .

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

step ~~8~~ 8.7 : else ~~go~~ set flag = 0

step 8.8 : Increament i by 1 and set ptr = ptr -> next

step 8.9 : check if flag = 0 , then point the element not found.

step 9 : Stop.

Output

Menu:

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

Enter choice : 1

Enter the element to be inserted: 2

Insertion is successful

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 elemat to be inserted: 10

Insertion is successful

Menu

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

Enter the choice: 2

Element deleted: 10

Menu

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

Enter the choice: 3

4 -> 2 -> null

Menu

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

Enter the choice : 4

Enter the item which is to be searched: 2

Item found at location : 2

Menu

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

Enter the choice : 5

3. circular queue operation

step1 : Start

step2 : declare the queue and other variables

step3 : Declare the functions for enque, deque, search & display

step4 : Read the choice from the user

step5 : If the user ~~so~~ choose the choice enque then read element to be inserted from the user and call the enqueue function by passing the value.

step5.1 : check if front == -1 && rear == -1, ~~the user choose the choi~~ then set front = 0, rear = 0 & set queue [rear] = element

step5.2 : ~~check if front ==~~ else if rear + 1 % max == front or front == rear + 1, then print queue overflow

step5.3 : else set rear = rear + 1 % max & set queue [rear] = element

step6 : If the user choice is the option dequeue then call the function dequeue.

step 6.1 : check if front == -1 & 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 & 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 + 1 % 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 queue

step 8.2 : check if item == queue[i] then
print item found and its position
and increament by 1

step 8.3: Check if c==0, then print item not found

step 9: End

output

```
Menu
1. Insert
2. Delete
3. Display
4. Search
5. Exit
Enter the choice : 1
Enter the number to insert: 10
Menu
1 Insert
2. Delete
3. Display
4. Search
5. Exit
Enter the choice : 1
Enter the number to insert: 20
Menu
1. Insert
2. Delete
3. Display
4. Search
5. Exit
```

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

Menu
1. Insert
2. Delete
3. Display
4. Search
5. Exit

Enter the choices: 3
10, 20, 30

Menu
1. Insert
2. Delete
3. Display
4. Search
5. Exit

Enter the choice: 4

Enter element which is to be searched: 30
Item found at location: 3

Menu
1. Insert
2. Delete

3. Display
4. Search
5. Exit

Enter the choice : 2
10 was deleted.

Menu
1. Insert
2. Delete
3. Display
4. Search
5. Exit

Enter the choice : 5

4. Doubly linked list operation.

step1: start

step2: declare a structure and related variables

step3: 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.

step4: 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 and temp -> next = null

~~step 4.2: set memory allocated to the node = temp, then set temp -> prev = null and temp ->~~

step 4.2: Read the value to be inserted to the node.

step 4.3: set temp -> n = data and increament 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 leginning then call the function to perform the insertion

step 6.1: Check if Read == 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 chose to perform insertion at the end of list, then call the function to perform the insertion at the end.

step 7.1: Check if head == null, the call the function to create 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 temp -> 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 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, the point "Empty list cannot 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 new node and then set temp->prev=temp2, temp->next=temp2->prev=temp.

temp2 -> next = temp

step 9: If the user choose to perform deletion operation in 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 needs to be deleted set temp2 = head

step 9.3: check if pos < 1 or pos >= 1 count + 1 the point position out of range

step 9.4: check if head == null then point the list is empty.

~~step 9.5: check if temp2 -> next == null then temp2 ->~~

step 9.5: while i < pos then temp2 = temp -> next and increament i by 1.

step 9.6: check if i = 1 then check if temp2 -> next == null, then print node deleted.

free temp2 and set temp2 -> head = null

step 9.7: check if temp2 -> next == null
then ~~temp2~~ temp2 -> prev -> next = null,
then free (temp2) and print node deleted

step 9.8: temp 2 -> next -> prev = temp2 -> prev
then check if i != 1, then temp 2 -> prev -> next
= temp2 -> next

step 9.9: check if i == 1, then head = temp2 -> next
then print node deleted then free temp2
and decreament count by 1.

~~step 10: set temp 2 = h~~

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

step 10.1: set temp 2 = h

step 10.2: check if temp 2 = null then print list
is empty

step 10.3: while temp 2 -> next != null then
print temp 2 -> n then temp 2 =
temp 2 -> 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 temp 2 = head

step 11.3: check if temp 2 == null ~~then~~ then print the list is empty

step 11.4: Read the value to be searched

step 11.5: While temp 2 != null, then check if temp 2 -> n == data then print element found at position count + 1.

step 11.6: Else set temp 2 = temp 2 -> next and increament count by 1

step 11.7: Print element not found in the list

step 12: End

Output

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

Enter choice: 3
Enter the position to be inserted: 2
Enter the value to node: 13

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

Enter choice: 5

Linked the element from beginning: 10 13 5

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

Enter choice: 7

Set Operations

step 1: START

step 2: Declare the necessary variables

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: i++

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 sets.

step 5.4: Repeat the step 5.5 to 5.7 until i<=n

step 5.5: c[i] = A[i] & B[i]

step 5.6: print c[i]

step 5.7: Increament 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, print cannot perform set difference.

step 6.3: ~~Est~~ Else read the element in both ~~side~~ sets.

step 6.4: ~~A~~ Repeat the step 6.5 to 6.8 until i<n.

step 6.5: check if A[i] = 0, the 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 : increament 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 : Increament i by 1.

Output

```
press 1 for union
press 2 for intersection
press 3 for subtraction
press 4 for exit
Enter choice: 1
Enter size of set1: 3
Enter the element of set 1:
1
2
3
Enter the size of set 2: 2
Enter the elements 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 your choice: 2
Enter the size of set1: 3
Enter elements of set1:
1
2
3
Enter the size of set2
2
Enter the elements of set2:
3
4
Intersection: 3
press 1 for union
press 2 for intersection
press 3 for subtraction
press 4 for exit
Enteryour choice: 3
Enter size of set1: 3
Enter elements of set1:
1
2
3
```

Enter size of set2 : 2

Enter elements of set2:

3

2

difference : 1

press 1 for union

Press 2 for intersection

press 3 for subtraction

Press 4 for exit

Enter choice : 4

Binary Search Tree

step 1: START

step 2: Declare a structure and 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 then read the value which is to be ~~ins~~ inserted.

step 5.1: Pass the value to the insert pointer and ~~also~~ also the root pointer.

step 5.2: check if !root then allocate memory for the root

Step 5.3: Set the value of 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 pass the root pointer and the item to the delete pointer

step 6.1: Check if not fte, 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 pasing the ~~it~~ left pointer 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 ~~fact~~ 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 to null

set p1 -> left, ptr -> left and free, ptr, return p2.

step6.9: Return ptr

step7: If the user choose to perform search operation the call the pointer to perform search operation.

step7.1: Declare the necessary pointers and variables

step7.2: Read the element to be searched

step7.3: While ptr check if item > ptr -> info, then ptr = ptr -> right

step7.4: Else if item < ptr -> info then ptr = ptr -> left.

~~step~~ step7.5: Else break.

step7.6: check if ptr then point 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 pointer.

Step 8.1: If root not equals to null recursively call the functions by passing root -> left.

Step 8.2: print root -> info

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

Step 9: 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

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 a 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 25 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 Program

Disjoint set operations

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 1

step 3.3: Set disrank[i] to 0.

step 3.4: Increament i 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: Increament 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: ~~Incro~~ Increament 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 & pass 2
variables x and y.

step 6.1: set ~~x set~~ xset to find(x)

step 6.2: set yset to find(y)

step 6.3: check if yset == xset, then return

step 6.4: check if disrank[xset] < disrank
[yset] then

step 6.5: set Yset = disparent[Yset]

~~step 6.6: set Yset = disparent[yset]~~

step 6.6: set -1 to disrank[xset]

step 6.7: else if check disrank[xset] >
disrank[yset]

step 6.8: set xset to disparent[yset]

step 6.9: set -1 to disrank[yset]

step 6.10: else disparent[yset] = xset

step 6.11 : Set disrank[xset] + 1 to disrank[xset]

step 6.12 : Set -1 to disrank[yset]

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 all 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. STOP

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 ? (1/0)
1
Menu
1. Union
2. Find
3. Display

Enter choice : 1
Enter elemet to perform union :
5
6
```

Do you wish to continue ? (1/0)

1.

Menu

1. Union
2. Find
3. Display

Enter choice :

3

Parent array :

3 1 2 3

Rank Array :

-1 0 0 1

Do you wish to continue ? (1/0)

0