1. **Merge two sorted arrays and store in a third array**

```
#include<stdio.h>
void main()
{
    int m,n,a[10],b[10],c[20],i,j,t,k=0;
    printf("Enter size of array a : ");
    scanf("%d",&m);
    printf("Enter array elements : ");
    for(i=0;i<m;i++)
    scanf("%d",&a[i]);
    printf("Enter size of array b : ");
    scanf("%d",&n);
    printf("Enter array elements : ");
    for(i=0;i<n;i++)
    scanf("%d",&b[i]);
    for(i=0;i<m;i++)
    for(j=i+1;j<m;j++)
    if(a[i]>a[j])
    {
            t=a[i];
            a[i]=a[j];
            a[j]=t;
    }
    for(i=0;i<n;i++)
    for(j=i+1;j<n;j++)
    if(b[i]>b[j])
    {
            t=b[i];
            b[i]=b[j];
```

```
        b[j]=t;
}
i=j=0;
while(i<m && j<n)
{
        if(a[i]<=b[j])
        {
                c[k]=a[i];
                i++;
                k++;
        }
        else
        {
                c[k]=b[j];
                j++;
                k++;
        }
}
while(i<m)
{
        c[k]=a[i];
        k++;
        i++;
}
while(j<n)
{
        c[k]=b[j];
        k++;
        j++;
```

```
    }
    printf("\nArray a : ");
    for(i=0;i<m;i++)
    printf("%d ",a[i]);
    printf("\nArray b : ");
    for(i=0;i<n;i++)
    printf("%d ",b[i]);
    printf("\nArray c : ");
    for(i=0;i<m+n;i++)
    printf("%d ",c[i]);
  }
```

2. Circular Queue - Add, Delete, Search

```
#include <stdio.h>
#include <stdlib.h>
int a[10], front = -1, rear = -1, n;
void insert();
void display();
void del();
void search();

int main()
{
   int ch;
   printf("Enter the size of the queue: ");
   scanf("%d", &n);
   while (1)
   {
      printf("\n\n1: Insertion");
```

```
    printf("\n2: Deletion");
    printf("\n3: Display");
    printf("\n4: Search");
    printf("\n5: Exit");
    printf("\nEnter your choice: ");
    scanf(" %d", &ch);

    switch (ch)
   {
        case 1:
            insert();
            break;
        case 2:
            del();
            break;
        case 3:
            display();
            break;
        case 4:
            search();
            break;
        case 5:
            printf("\nPress any key to exit..");
            exit(0);
        default:
            printf("\nInvalid choice");
    }
}
return 0;
```

```
}

void insert()
 {
    int x;
    if ((front == 0 && rear == n - 1) || (front == rear + 1))
   {
        printf("Queue is full");
    } else
    {
        printf("Enter the element to insert: ");
        scanf("%d", &x);
        if (front == -1 && rear == -1)
            front = rear = 0;
        else if (rear == n - 1 && front != 0)
            rear = 0;
        else
            rear = (rear + 1) % n;
        a[rear] = x;
    }
}

void display()
{
    int i;
    printf("Front = %d\nRear = %d\n", front, rear);
    if (front == -1)
        printf("\nQueue is empty");
    else if (front <= rear)
```

```
    {
        for (i = front; i <= rear; i++)
            printf("%d ", a[i]);
    }
    else
      {
        for (i = front; i < n; i++)
            printf("%d ", a[i]);
        for (i = 0; i <= rear; i++)
            printf("%d ", a[i]);
     }
}

void del()
{
    if (front == -1)
        printf("\nQueue is empty");
    else
   {
        printf("Deleted element: %d", a[front]);
        if (front == rear)
            front = rear = -1;
        else
      {
            if (front == n - 1)
                front = 0;
            else
                front += 1;
        }
```

```
    }
}

void search()
{
    int x,i,j;
    printf("Enter the element to search : ");
    scanf("%d",&x);
    if(front<=rear)
     {
        int f1=0;
        for(i=front;i<=rear;i++)
          {
            if(a[i]==x)
             {
                printf("Element found at position %d",i);
                f1=1;
                break;
             }
         }
        if(f1==0)
        printf("Element not found");
    }
    else
    {
        int f=0;
        for(i=front,j=1;i<n;i++,j++)
         {
            if(a[i]==x)
```

```
            {
                f=1;
                printf("Element found at position : %d",j);
                break;
            }
        }
        if(f==0)
        {
            int f2=0;
            for(i=0;i<=rear;i++)
            {
                if(a[i]==x)
                {
                    printf("Element found at position : %d",i+n-1);
                    f2=1;
                    break;
                }
            }
            if(f2==0)
            printf("Element not found");
        }
    }
}
```

3. Singly Linked Stack - Push, Pop, Linear Search,Display

```
 #include<stdio.h>
#include<stdlib.h>
struct node
{
  int data;
 struct node *next;
} ;
```

```
struct node *start;
void push()
{
 int x;
 struct node *ptr;
 ptr=malloc(sizeof(struct node));
 if(ptr==NULL)
 {
        printf("\nCan't push element");
 }
 else
 {
        printf("\nEnter the value : ");
        scanf("%d",&x);
        if(start==NULL)
        {
                ptr->data=x;
                ptr->next=NULL;
                start=ptr;
        }
        else
        {
                ptr->data=x;
                ptr->next=start;
                start=ptr;
        }
 }
}

void pop()
{
 int x;
 struct node *ptr;
 if(start==NULL)
 {
        printf("\nUnderflow");
 }
 else
 {
        x=start->data;
        ptr=start;
        start=start->next;
        free(ptr);
```

```
        printf("Element popped - %d",x);
 }
}


void traverse()
{
   struct node* temp;
   if (start == NULL)
      printf("\nList is empty\n");
   else
   {
      temp = start;
      printf("the list is\n");
      while (temp != NULL)

      {
         printf(" %d -->", temp->data);
         temp = temp->next;
      }
   }
}

void search()
{
 int i=1,f=0,x;
 struct node *ptr;
 ptr=start;
 if(ptr==NULL)
 {
        printf("\nStack is empty");
 }
 else
 {
        printf("\nEnter element : ");
        scanf("%d",&x);
        while(ptr!=NULL) {
                if(ptr->data==x)
                {
                        f=1;
                        break;
                }
                i++;
```

```
                ptr=ptr->next;
        }
        if(f==0)
                printf("\nItem not found");
        else
                printf("\nItem found at position %d",i);
 }
}
void main()
{
 int ch=0;

 while(ch!=5)
 {
        printf("\n\n1:Push");
        printf("\n2:Pop");
        printf("\n3:Linear search");
        printf("\n4:Display");
        printf("\n5:Exit");
        printf("\nEnter your choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:push();
                break;
                case 2:pop();
                break;
                case 3:search();
                break;
                case 4:traverse();
                break;
                case 5:exit(0);
                break;
                default:printf("\nInvalid choice");
        }
 }
}
```

4. **Singly Linked List Insertion,Deletion**

```
 #include <stdio.h>
#include <stdlib.h>
struct node
{
```

```
    int info;
    struct node* link;
};
struct node* start = NULL;

void traverse()
{
    struct node* temp;

    if (start == NULL)
        printf("\nList is empty\n");
    else
    {
        temp = start;
        printf("the list is\n");
        while (temp != NULL) {
            printf(" %d -->", temp->info);
            temp = temp->link;
        }
    }
}

 void insertAtFront()
{
    int data;
    struct node* temp;
    temp = malloc(sizeof(struct node));
    printf("\nEnter number to be inserted : ");
    scanf("%d", &data);
    temp->info = data;
    temp->link = start;
    start = temp;
}

void insertAtEnd()
{
    int data;
    struct node *temp, *head;
    temp = malloc(sizeof(struct node));
    printf("\nEnter number to be inserted : ");
    scanf("%d", &data);
    temp->link = 0;
    temp->info = data;
```

```
    head = start;
    while (head->link != NULL)
    {
        head = head->link;
    }
    head->link = temp;
}

void insertAtPosition()
{
    struct node *temp, *newnode;
    int pos, data, i = 1;
    newnode = malloc(sizeof(struct node));
    printf("\nEnter position and data :");
    scanf("%d %d", &pos, &data);
    temp = start;
    newnode->info = data;
    newnode->link = 0;
    while (i < pos - 1)
    {
        temp = temp->link;
        i++;
    }
    newnode->link = temp->link;
    temp->link = newnode;
}

void deleteFirst()
{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    else
    {
        temp = start;
        start = start->link;
        free(temp);
    }
}

void deleteEnd()
{
    struct node *temp, *prevnode;
```

```
    if (start == NULL)
        printf("\nList is Empty\n");
    else
    {
        temp = start;
        while (temp->link != 0)
        {
            prevnode = temp;
            temp = temp->link;
        }
        free(temp);
        prevnode->link = 0;
    }
}

void deletePosition()
{
    struct node *temp, *position;
    int i = 1, pos;
    if (start == NULL)
        printf("\nList is empty\n");
    else
    {
        printf("\nEnter position : ");
        scanf("%d", &pos);
        position = malloc(sizeof(struct node));
        temp = start;
        while (i < pos - 1)
        {
            temp = temp->link;
            i++;
        }
        position = temp->link;
        temp->link = position->link;
        free(position);
    }
}

void search()
{
    int found = -1, key;
    struct node *tr = start;
    if (start == NULL)
```

```
    {
        printf("Linked list is empty\n");
    }
    else
    {
        printf("\nEnter the element you want to search: ");
        scanf("%d", &key);
        while (tr != NULL)
        {
            if (tr->info == key)
            {
                found = 1;
                break;
            }
            else
            {
                tr = tr->link;
            }
        }

        if (found == 1)
        {
            printf("Yes, %d is present in the linked list.\n",key);
        }
        else
         {
            printf("No, %d is not present in the linked list.\n",key);
        }
    }
}

void main()
{
    int choice;
    while (1)
{
        printf("\n\t1  To see list\n");
        printf("\t2  For insertion at starting\n");
        printf("\t3  For insertion at end\n");
        printf("\t4  For insertion at any position\n");
        printf("\t5  For deletion of first element\n");
        printf("\t6  For deletion of last element\n");
        printf("\t7  For deletion of element at any position\n");
```

```
        printf("\t8 Search an element in linked list\n");
        printf("\t9 To exit\n");
        printf("\nEnter Choice :\n");
        scanf("%d", &choice);

        switch (choice)
       {
        case 1:
           traverse();
           break;
        case 2:
           insertAtFront();
           break;
        case 3:
           insertAtEnd();
           break;
        case 4:
           insertAtPosition();
           break;
        case 5:
           deleteFirst();
           break;
        case 6:
           deleteEnd();
           break;
        case 7:
           deletePosition();
           break;

        case 8:
           search();
           break;
        case 9:
           exit(1);
           break;
        default:
           printf("Incorrect Choice\n");
        }
    }
}
```

5. Implement all the operations of doubly linked list

```
#include<stdio.h>

#include<stdlib.h>
struct node
{
   int data;
   struct node *prev;
   struct node *next;
};
struct node *start;
void beginsert()
{
   struct node *ptr;
   int x;
   ptr=(struct node*)malloc(sizeof(struct node*));
   if(ptr==NULL)
{
      printf("\nOverflow");
   }
   else
   {
      printf("\nEnter value : ");
      scanf("%d",&x);
      if(start==NULL)
      {
         ptr->data=x;
         ptr->prev=NULL;
         ptr->next=NULL;
         start=ptr;
      }
      else
     {
         ptr->data=x;
         ptr->prev=NULL;
         ptr->next=start;
         start->prev=ptr;
         start=ptr;
      }
   }
}
void lastinsert()
{
   struct node *ptr,*temp;
```

```
    int x;
    ptr=(struct node*)malloc(sizeof(struct node*));
    if(ptr==NULL)
   {
       printf("\nOverflow");
    }
    else
 {
       printf("\nEnter value : ");
       scanf("%d",&x);
       ptr->data=x;
       if(start==NULL)
      {
          ptr->next=NULL;
          ptr->prev=NULL;
          start=ptr;
       }
       else
       {
          temp=start;
          while(temp->next!=NULL) {
             temp=temp->next;
          }
          temp->next=ptr;
          ptr->prev=temp;
          ptr->next=NULL;
       }
    }
}
void posinsert()
{
    int pos,i,x;
    struct node *ptr,*temp;
    ptr=(struct node*)malloc(sizeof(struct node*));
    if(ptr==NULL)
    {
       printf("\nOverflow");
    }
    else
  {
       printf("\nEnter value : ");
       scanf("%d",&x);
       ptr->data=x;
       printf("\nEnter the position : ");
       scanf("%d",&pos);
       temp=start;
```

```
        if(pos==1)
         {
           if(start==NULL) {
              ptr->data=x;
              ptr->prev=NULL;
              ptr->next=NULL;
              start=ptr;
           }
           else
           {
              ptr->data=x;
              ptr->prev=NULL;
              ptr->next=start;
              start->prev=ptr;
              start=ptr;
           }
        }
        else
        {
           for(i=1;i<=pos;i++)
            {
              temp=temp->next;
              if(temp==NULL)
              {
                 printf("\nCan't insert");
              }
           }
           ptr->next=temp->next;
           temp->next=ptr;
        }
    }
}
void begdel()
 {
    int x;
    struct node *ptr;
    if(start==NULL) {
       printf("\nList is empty");
    }
    else
   {
       ptr=start;
       start=ptr->next;
       start->prev=NULL;
       x=ptr->data;
       free(ptr);
```

```
        printf("\n%d deleted",x);
    }
}
void lastdel()
 {
    int x;
    struct node *ptr,*ptr1;
    if(start==NULL)
  {
        printf("\nList is empty");
    }
    else if(start->next==NULL)
    {
        x=start->data;
        start=NULL;
        free(start);
        printf("\n%d deleted",x);
    }
    else
  {
        ptr=start;
        while(ptr->next!=NULL)
   {
            ptr1=ptr;
            ptr=ptr->next;
        }
        ptr1->next=NULL;
        ptr->prev=NULL;
        x=ptr->data;
        free(ptr);
        printf("\n%d deleted",x);
    }
}
void posdel()
 {
    struct node *ptr,*ptr1;
    int pos,i,x;
    ptr=start;
    if(ptr==NULL)
 {
        printf("\nList is empty");
    }
    else
   {
        printf("\nEnter position : ");
        scanf("%d",&pos);
```

```
        if(pos==1) {
            ptr=start;
            start=ptr->next;
            start->prev=NULL;
            x=ptr->data;
            free(ptr);
            printf("\n%d deleted",x);
        }
        else
     {
            for(i=1;i<pos;i++)
          {
                ptr1=ptr;
                ptr=ptr->next;
                if(ptr==NULL)
               {
                    printf("\nCan't delete");
                    return;
                }
            }
            ptr1->next=ptr->next;
            (ptr->next)->prev=ptr1;
            x=ptr->data;
            free(ptr);
            printf("\n%d deleted",x);
        }
    }
}
void search()
{
    struct node *ptr;
    int x,i=0,f;
    ptr=start;
    if(ptr==NULL)
   {
        printf("\nList is empty");
    }
    else
   {
        printf("\nEnter element to search : ");
        scanf("%d",&x);
        while(ptr!=NULL)
        {
            if(ptr->data==x)
             {
                printf("\nItem found at position %d",i+1);
```

```
            f=0;
            break;
          }
          else
          {
            f=1;
          }
          i++;
          ptr=ptr->next;
        }
        if(f==1)
        {
          printf("\nItem not found");
        }
    }
}

void display()
 {
    struct node *ptr;
    ptr=start;
    if(ptr==NULL)
 {
        printf("\nList is empty");
    }
    else {
        while(ptr!=NULL)
    {
            printf("%d-> ",ptr->data);
            ptr=ptr->next;
        }
        printf("null");
    }
}
void main()
 {
    int ch;
    while(ch!=9)
  {
        printf("\n\n1:Insert at beginning");
        printf("\n2:Insert at last");
        printf("\n3:Insert at position");
        printf("\n4:Delete from beginning");
        printf("\n5:Delete from last");
        printf("\n6:Delete from position");
        printf("\n7:Search");
```

```
        printf("\n8:Display");
        printf("\n9:Exit");
        printf("\nEnter your choice : ");
        scanf("%d",&ch);
        switch(ch)
    {
            case 1:beginsert();
            break;
            case 2:lastinsert();
            break;
            case 3:posinsert();
            break;
            case 4:begdel();
            break;
            case 5:lastdel();
            break;
            case 6:posdel();
            break;
            case 7:search();
            break;
            case 8:display();
            break;
            case 9:exit(0);
            break;
            default:printf("\nInvalid choice");
        }
    }
}
```

6) Binary Search Trees- Insertion, Deletion, Search and Traverse

```
#include<stdio.h>
#include<stdlib.h>

struct node
{
 struct node *left;
 struct node *right;
 int data;
};
struct node *root;

struct node* newNode(value)
{
 struct node *newnode = malloc(sizeof(struct node));
 newnode->data = value;
```

```
 newnode->left=NULL;
 newnode->right=NULL;
 return newnode;
}

struct node* insert(struct node* root,int value)
{
  if(root == NULL){
          return newNode(value);
  }
  else if(value == root->data)
{
          printf("Same data can't be stored");
  }
  else if(value>root->data)
{
          root->right = insert(root->right,value);
  }
  else if(value<root->data)
{
          root->left = insert(root->left,value);
  }
  return root;
}

// Inorder traversal
void inorderTraversal(struct node* root)
 {
  if (root == NULL) return;
  inorderTraversal(root->left);
  printf("%d ->", root->data);
  inorderTraversal(root->right);
}

// Preorder traversal
void preorderTraversal(struct node* root)
 {
  if (root == NULL) return;
  printf("%d ->", root->data);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}

// Postorder traversal
void postorderTraversal(struct node* root)
 {
```

```
  if (root == NULL) return;
  postorderTraversal(root->left);
  postorderTraversal(root->right);
  printf("%d ->", root->data);
}

struct node* search(struct node* root, int key)
 {
  if (root == NULL)
    printf("\nNot FOUND!\n");
  else if (root->data == key)
    printf("\nFOUND!\n");
  else
 {
          if (root->data < key)
    return search(root->right, key);
  return search(root->left, key);
  }
}

struct node* minValueNode(struct node* node)
{
    struct node* current = node;

    /* loop down to find the leftmost leaf */
    while (current && current->left != NULL)
        current = current->left;

    return current;
}

struct node* deleteNode(struct node* root, int key)
{
    if (root == NULL)
        return root;

    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        // node with only one child or no child
        if (root->left == NULL) {
            struct node* temp = root->right;
            free(root);
            return temp;
```

```
        }
        else if (root->right == NULL)
 {
            struct node* temp = root->left;
            free(root);
            return temp;
        }
        // node with two children:
        // Get the inorder successor
        // (smallest in the right subtree)
        struct node* temp = minValueNode(root->right);

        // Copy the inorder
        // successor's content to this node
        root->data = temp->data;

        // Delete the inorder successor
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

void main()
{
 int opt;
 int value,searchv,key;
 do{
          printf("\n1)Create Root Node \n2)Insert Node\n3)Search\n");
          printf("4)inorderTraversal      \n5)preorderTraversal      \n6)postorderTraversal
\n7)Delete \n8)Quiet \n");
          printf("Choose Option :: ");
          scanf("%d",&opt);
          switch(opt)
          {
                    case 1:
                              printf("\nEnter a number : ");
                              scanf("%d",&value);
                              root = newNode(value);
                              break;
                    case 2:
                              printf("\nEnter a number : ");
                              scanf("%d",&value);
                              root = insert(root,value);
                              break;
                    case 3:
                              printf("\nEnter a number : ");
```

```
                scanf("%d",&searchv);
                search(root,searchv);
                break;
        case 4:
                printf("\n.....\n");
                inorderTraversal(root);
                printf("\n.....\n");
                break;
        case 5:
                printf("\n.....\n");
                preorderTraversal(root);
                printf("\n.....\n");
                break;
        case 6:
                printf("\n.....\n");
                postorderTraversal(root);
                printf("\n.....\n");
                break;
        case 7:
                printf("\nEnter a number to be deleted : ");
                scanf("%d",&key);
                deleteNode(root,key);
                break;
        defualt:
                printf("Invalid option!");
    }
 }while(opt!=8);
}
```