INMANTEC INSTITUTIONS
Since: 1995

DATA STRUCTURE AND ALGORITHM

Name	Prakash
Roll no.	205028
Course	BCA
Year	2nd
Semester	3rd
Section	A

Submitted By:

Submitted To:

Date of submission:

INDEX

DATA STRUCTURE AND ALGORITHMS LAB FILE

プラカシュ

```
//DSAL_1_Stack_Program_using_Array_With_Operation_Push, POP, Display
#include<iostream>
#define MAX 100
using namespace std;
class stackUsingArray
{
    private:
        int top;

    public:
        int stack[MAX] = {0};
        void push(int x, int n);
        void pop();
        void displayStack();
};

void stackUsingArray::push(int x, int n)
{
    int item = x;
    if(top == n)
    {
        cout << "OverFlow: Stack is Full!";
    }
    else
    {
        top= top+1;
        stack[top] = item;
        cout << "Element Pushed!";
    }
}

void stackUsingArray::pop()
{
    if(stack[top] == 0)
    {
        cout << "UnderFlow: Stack is empty!";
```

```
    }
    else
    {
        top = top-1;
        cout << "Element Poped ";
    }
}

void stackUsingArray::displayStack()
{
    int i = 1;
    if(top==0)
    {
        cout << "Stack is empty!";
    }
    for(i = top; i>0; i--)
    {
        cout << stack[i]<<'\n';
    }
}
int main()
{
    int n= 0, choice = 0, pushElement = 0;
    stackUsingArray stack1;
    cout << "Enter The no. of elements in
stack(<100): ";
    cin >> n;
    while(choice != 4)
    {
        cout << "\n====Stack Operation====\n";
        cout <<
"\n1.Push\n2.Pop\n3.Show\n4.Exit\n";
        cout << "Chose one Option by their no.: ";
        cin >> choice;
        if(choice == 1)
        {
            cout << "Enter The no. to push ";
```

```
            cin >> pushElement ;
        }
        switch(choice)
        {
            case 1:
            {
                stack1.push(pushElement, n);
                break;
            }
            case 2:
            {
                stack1.pop();
                break;
            }
            case 3:
            {
                stack1.displayStack();
                break;
            }
            case 4:
            {
                cout << "Exiting....";
                break;
            }
            default:
            {
                cout << "Please Enter valid choice
";
            }
        };
    }
    return 0;
}
```

//DSAL_2_Queue_Program_using_Array_With_Operation_insert_delete_display

```
#include<iostream>
#define MAX 5

using namespace std;

class QueueUsingArray
{
    private:
        int front = -1, rear = -1;
    public:
        int queue[MAX] = {0};
        void insertElement(int x );
        void deleteElement();
        void displayQueue();

};

void QueueUsingArray::insertElement(int x )
{
    int element = x;
    if(rear == MAX-1)
    {
        cout << "OverFlow: Queue is Full!";
        return;
    }
    if(front == -1 && rear == -1)
    {
        front=0;
        rear=0;
    }
    else
    {
        rear = rear+1;
    }

    queue[rear] = element;
    cout << "Element Inserted!";
}

void QueueUsingArray::deleteElement()
{
    int val;

    if(front == -1 || front > rear)
```

```
    {
        cout << "UnderFlow: Queue is empty!";
        return;
    }
    else
    {
        val = queue[front];

        if(front == rear)
        {
            front = -1;
            rear = -1;
        }
        else
        {
            front = front + 1;
        }
        cout << "Element deleted";
    }
}

void QueueUsingArray::displayQueue()
{
    int i = 0; // = 1;

    if(rear == -1)
    {
        cout << "Queue is empty!";
    }
    else
    {
        cout<<"Queue elements: ";

        for(i=front;i<=rear;i++)
        {
            cout << queue[i] << " ";
        }
    }
}

int main()
{
    int choice = 0, insertElement = 0;

    QueueUsingArray Queue1;
```

```
    // cout << "Enter The no. of elements in Queue(<100): ";
    // cin >> n;

    while(choice != 4)
    {
        cout << "\n====Queue Operation====\n";
        cout << "\n1.insert\n2.Delete\n3.Display\n4.Exit\n";
        cout << "Chose one Option by their no.: ";

        cin >> choice;

        if(choice == 1)
        {
            cout << "Enter The element to push ";
            cin >> insertElement ;
        }
        switch(choice)
        {
            case 1:
            {
                Queue1.insertElement(insertElement);
                break;
            }
            case 2:
            {
                Queue1.deleteElement();
                break;
            }
            case 3:
            {
                Queue1.displayQueue();
                break;
            }
            case 4:
            {
                cout << "Exiting....";
                break;
            }
            default:
            {
                cout << "Please Enter valid choice ";
            }
        };
    }
    return 0;
}
```

```
//DSAL_3_Circular_Queue_Program_using_Array_With_Operation_insert_delet
e_display
#include<iostream>
#define MAX 5

using namespace std;

class CirQueueUsingArray
{
    private:
        int front = -1, rear = -1;

    public:
        int queue[MAX] = {0};


        void insertElement(int x );
        void deleteElement();
        void displayCirQueue();

};

void CirQueueUsingArray::insertElement(int x )
{
    int element = x;
    if((rear+1)%MAX == front)
    {
        cout << "OverFlow: Circular Queue is Full!";
        return;
    }
    if(front == -1 && rear == -1)
    {
        front=0;
        rear=0;

    }
    else if(rear == MAX-1 && front != 0)
    {
        rear = 0;
    }
    else
    {
        rear = (rear+1)%MAX;

    }
```

```
    queue[rear] = element;
    cout << "Element Inserted!";
}

void CirQueueUsingArray::deleteElement()
{
    int val;

    if(front == -1 || front > rear)
    {
        cout << "UnderFlow: Circular Queue is empty!";
        return;
    }
    else
    {
        val = queue[front];

        if(front == rear)
        {
            front = -1;
            rear = -1;
        }
        else if(front == MAX-1)
        {
            front = 0;
        }
        else
        {
            front = front + 1;
        }

        cout << "Element deleted";
    }

}

void CirQueueUsingArray::displayCirQueue()
{
    int i=front;
    if(front==-1)
    {
        cout<<"\n Circular Queue is empty..";
    }
    else
    {
        cout<<"\nElements in a circular Queue are :";
```

```
        if(front<=rear)
        {
            while(i <= rear)
            {
                cout<< queue[i]<<" ";
                i++;
            }

        }
        else
        {
            while(i<=MAX-1)
            {
                cout<< queue[i]<<" ";
                i++;
            }
            while(i <= rear)
            {
                cout<< queue[i]<<" ";
                i++;
            }
            i=0;

        }
    }
}

int main()
{
    int choice = 0, insertElement = 0;

    CirQueueUsingArray Queue1;

    // cout << "Enter The no. of elements in Queue(<100): ";
    // cin >> n;

    while(choice != 4)
    {
        cout << "\n====Cirular Queue Operation====\n";
        cout << "\n1.insert\n2.Delete\n3.Display\n4.Exit\n";
        cout << "Chose one Option by their no.: ";

        cin >> choice;

        if(choice == 1)
        {
```

```
            cout << "Enter The element to push ";
            cin >> insertElement ;
        }
        switch(choice)
        {
            case 1:
            {
                Queue1.insertElement(insertElement);
                break;
            }
            case 2:
            {
                Queue1.deleteElement();
                break;
            }
            case 3:
            {
                Queue1.displayCirQueue();
                break;
            }
            case 4:
            {
                cout << "Exiting....";
                break;
            }
            default:
            {
                cout << "Please Enter valid choice ";
            }
        };
    }
    return 0;
}
```

//DSAL_4_Linked_List_Program_With_Operations

```
#include<iostream>
#include<vector>
using namespace std;

class node
{
    public:
        int v;
        node *next;
        node()
        {
            next = NULL;
        }
};

class LinkedList
{
    node *head;
    public:

        LinkedList()
        {
            head = NULL;
        }

        void insert_at_beginning(int v)
        {
            node *temp = new node();
            temp->v = v;
            temp->next = head;
            head = temp;
        }

        void insert_at_end(int v)
        {
            node *temp = new node();
            temp->v = v;
            if (head == NULL)
            {
                head = temp;
            }
            else{
                node *ptr = head;
```

```
        while (ptr->next != NULL)
        {
            ptr = ptr->next;
        }
        ptr->next = temp;
    }
}

void insert_at_given_position(int v, int p)
{
    node *temp = new node();
    temp->v = v;
    if (p == 0)
    {
        temp->next = head;
        head = temp;
    }
    else
    {
        node *ptr = head;

        while(p>1) {
            ptr = ptr->next;
            --p;
        }

        temp->next = ptr->next;
        ptr->next = temp;
    }
}

void delete_at_beginning()
{
    if (head == NULL)
    {
        cout<<"List is Empty"<<"\n";
    }
    else{
        cout<<"Element Deleted: "<<head->v<<"\n";

        node *temp = head;

        head = head->next;

        delete(temp);
    }
```

```
}

void delete_at_end()
{
    if (head == NULL)
    {
        cout<<"List is Empty"<<"\n";
    }
    else if (head->next == NULL)
    {

        cout<<"Element Deleted: "<<head->v<<"\n";
        delete(head);
        head = NULL;
    }
    else
    {

        node *temp = head;

        while (temp->next->next != NULL) {
            temp = temp->next;
        }

        cout<<"Element Deleted: "<<temp->next->v<<"\n";
        // delete last node
        delete(temp->next);

        temp->next = NULL;
    }

}

void delete_at_given_position(int p)
{
    if (head == NULL)
    {
        // if list is empty do nothing
        cout<<"List is Empty"<<"\n";
    }
    else
    {
        node *temp, *ptr;
        if (p == 0)
        {
```

```
                    cout<<"Element Deleted: "<<head->v<<"\n";
                    ptr = head;
                    head = head->next;
                    delete(ptr);
                }
                else
                {

                    temp = ptr = head;
                    while(p>0){
                        --p;
                        temp = ptr;
                        ptr = ptr->next;
                    }
                    cout<<"Element Deleted: "<<ptr->v<<"\n";

                    temp->next = ptr->next;

                    free(ptr);
                }
            }

        }

        void display()
        {
            if (head == NULL)
            {
                cout<<"List is empty"<<"\n";
            }
            else
            {
                node *temp = head;
                cout<<"Linked List: ";
                while (temp != NULL)
                {
                    cout<<temp->v<<"->";
                    temp = temp->next;
                }
                cout<<"NULL"<<"\n";
            }
        }

};
```

```
int main()
{

    cout<<"1 to Insert at the beginning";
    cout<<"\n2 to Insert at the end";
    cout<<"\n3 to Insert at mid";
    cout<<"\n4 to Delete from beginning";
    cout<<"\n5 to Delete from the end";
    cout<<"\n6 to Delete from mid";
    cout<<"\n7 to Display";
    cout<<"\n0 to Exit";

    int choice,v,p;
    LinkedList ll;
    do {
        cout<<"\nEnter Your Choice: ";
        cin>>choice;
        switch (choice)
        {
            case 1:
                cout<<"Enter Element: ";
                cin>>v;
                ll.insert_at_beginning(v);
                break;

            case 2:
                cout<<"Enter Element: ";
                cin>>v;
                ll.insert_at_end(v);
                break;

            case 3:
                cout<<"Enter Element: ";
                cin>>v;
                cout<<"Enter Position ( zero-indexed ): ";
                cin>>p;
                ll.insert_at_given_position(v,p);
                break;

            case 4:
                ll.delete_at_beginning();
                break;

            case 5:
                ll.delete_at_end();
                break;
```

```
            case 6:
                cout<<"Enter Position ( zero-indexed ): ";
                cin>>p;
                ll.delete_at_given_position(p);
                break;

            case 7:
                ll.display();
                break;

        }
    } while (choice != 0);

}
```

```
//DSAL_5_Doubly_Linked_List_With_Operations

#include<iostream>
using namespace std;

template<typename T>class Node
{
    private:
        T data;
        Node<T>* next;
        Node<T>* prev;
        template<typename U>friend class LinkedList;
    public:
        Node()
        {
            this->next = NULL;
            this->prev = NULL;
        }
};

template<typename T>class LinkedList
{
    private:
        Node<T>* head;
    public:
        LinkedList()
        {
            this->head = NULL;
        }

        void add(T item)
        {
            Node<T>* node = new Node<T>[1];
            node->data = item;
            if(head == NULL)
            {
                head = node;
                cout<<"new node added(firstnode) !"<<endl;
                return;
            }
            Node<T>* temp = head;
            Node<T>* prev;
            while(temp->next != NULL)
            {
                prev = temp;
                temp = temp->next;
```

```
            }
            temp->next = node;
            temp->prev = prev;
            cout<<"new node added at back!"<<endl;
        }

        void addFront(T item)
        {
            Node<T>* node = new Node<T>[1];
            node->data = item;
            if(head == NULL)
            {
                head = node;
                cout<<"new node added(firstnode) !"<<endl;
                return;
            }
            head->prev = node;
            node->next = head;
            head = node;
            cout<<"new node added at front !"<<endl;
        }

        void add(int index, T item)
        {
            if(index > length() || index < 0)
            {
                cout<<"index out of bound !"<<endl;
                return;
            }
            Node<T>* node = new Node<T>[1];
            node->data = item;
            int count = 0;
            Node<T>* temp = head;
            while(temp != NULL && count < index)
            {
                if(count == index-1)
                {
                    if(temp->next != NULL)
                    {
                        node->next = temp->next;
                    }
                    temp->next = node;
                    node->prev = temp;
                    cout<<"new node added at index "<<index<<"
!"<<endl;
                    break;
```

```
        }
        count++;
        temp = temp->next;
    }

}
int length(){
    int len = 0;
    Node<int>* temp = head;
    while(temp != NULL){
        len++;
        temp = temp->next;
    }
    return len;
}

void displayAll()
{
    if(head == NULL)
    {
        cout<<"linked list is empty"<<endl;
        return;
    }
    cout<<endl<<"----link list items------"<<endl;
    Node<T>* temp = head;
    while(temp != NULL)
    {
        cout<<temp->data<<" | ";
        temp = temp->next;
    }
    cout<<endl<<"-------------------------"<<endl;
}
void remove(int index)
{
    if(head == NULL)
    {
        cout<<"linked list is empty !"<<endl;
        return;
    }
    if(index >= length() || index < 0)
    {
        cout<<"index out of bound !"<<endl;
        return;
    }
    if(index == 0)
    {
```

```
                //removeFront();
                cout<<"item removed at index "<<index<<endl;
                return;
            }
            int count = 0;
            Node<T>* temp = head;
            while(temp != NULL)
            {
                if(count == index - 1)
                {
                    temp->next = temp->next->next;
                    cout<<"item removed at index "<<index<<endl;
                    break;
                }
                count++;
                temp = temp->next;
            }
        }
};

int main()
{
    LinkedList<int> list;
    int ch, item, index;
    bool quit = false;
    do{
        cout<<"==========Doubly LinkedList==========="<<endl;
        cout<<"select one of the option :"<<endl;
        cout<<"1: insert back"<<endl;
        cout<<"2: insert front"<<endl;
        cout<<"3: insert at index"<<endl;
        cout<<"4: display items"<<endl;
        cout<<"5: delete at index"<<endl;
        cout<<"6: exit"<<endl;
        cin>>ch;

        switch (ch)
        {
        case 1:
            cout<<"enter item to insert:"<<endl;
            cin>>item;
            list.add(item);
            break;
        case 2:
            cout<<"enter item to insert:"<<endl;
            cin>>item;
```

```
            list.addFront(item);
            break;
        case 3:
            cout<<"enter item to insert:"<<endl;
            cin>>item;
            cout<<"enter index:"<<endl;
            cin>>index;
            list.add(index, item);
            break;
        case 4:
            list.displayAll();
            break;
        case 5:
            cout<<"enter index:"<<endl;
            cin>>index;
            list.remove(index);
            break;
        case 6:
            quit = true;
            break;
        default:
            cout<<"invalid selection"<<endl;
            break;
        }
    }while(!quit);
    return 0;
}
```

//DSAL_6_Circular_Linked_List_With_Operations

```
#include<iostream>
using namespace std;

struct Node
{
    int data;
    Node *next;
}*tail;

class Circular_LinkedList
{
public:
    void Insert_at_Front(int n);
    void Insert_at_End(int n);
    void Delete_at_Front();
    void Display();
};

int main()
{
    int val = 0;
    int choice;

    Circular_LinkedList object;

    tail=NULL;

    while(1)
    {
        cout<<"\n=========Enter Value to Be inserted============\n";
        cout<<"\n\t1.Insert at Front\n";
        cout<<"\t2.Insert at End\n";
        cout<<"\t3.Delete at Front\n";
        cout<<"\t4.Display Nodes\n";
        cout<<"\t5.Exit\n";
        cout<<" Enter you Choice: \n";
        cin>>choice;
        switch(choice)
        {
            case 1:
                cout<<"Enter Value to Be inserted: ";
                cin>> val;

                object.Insert_at_Front(val);
```

```
                cout<<"Value " <<val <<" inserted at Front.\n";
                break;
            case 2:
                cout<<"Enter Value to Be inserted: ";
                cin>> val;

                object.Insert_at_End(val);
                cout<<"Value " <<val <<" inserted at End.\n";
                break;
            case 3:
                object.Delete_at_Front();
                cout<<"Value " <<val <<" Deleted at Front.\n";
                break;
            case 4:
                object.Display();
                break;
            case 5:
                exit(0);
            default:
                cout<<"invalid Choice!\n";
        }
    }
return 0;
}

void Circular_LinkedList::Insert_at_Front(int n)
{
    Node *temp;
    temp=new Node;
    temp->data=n;
    temp->next=NULL;
    if(tail==NULL)
    {
        tail=temp;
        tail->next=tail;
    }
    else
    {
        temp->next=tail->next;
        tail->next=temp;
    }
}

void Circular_LinkedList::Insert_at_End(int n)
{
    Node *temp;
```

```
    temp=new Node;
    temp->data=n;
    temp->next=NULL;

    if(tail==NULL)
    {
        temp->next=temp;
        tail=temp;
    }
    else
    {
        temp->next=tail->next;
        tail->next=temp;
        tail=temp;
    }
}

void Circular_LinkedList::Display()
{
    Node *front;
    front=tail->next;

    if(tail==NULL)
    {
        cout<<"Empty List: \n";
    }

    cout<<"\tNodes are:\t";

    while(front!=tail)
    {
        cout<<front->data<<" --> ";
        front=front->next;
    }

    if(front==tail)
    {
        cout<<front->data;
    }

    delete front;
    cout<<endl;
}

void Circular_LinkedList::Delete_at_Front()
{
```

```
    Node *temp=new Node;
    temp=tail->next;
    if(tail==NULL)
    {
        cout<<"Empty List\n";
        return;
    }
    if(temp==tail)
    {
        delete temp;

    }
    else
    {
        tail->next=temp->next;
        delete temp;
    }
}
```

```
//DSAL_7_Tree_Implementation_With_Operations

#include <iostream>

using namespace std;


struct node{
    int value;
    node *left;
    node *right;
};

class btree{
public:
    btree();
    ~btree();

    void insert(int key);
    node *search(int key);
    void destroy_tree();
    void inorder_print();
    void postorder_print();
    void preorder_print();

private:
    void destroy_tree(node *leaf);
    void insert(int key, node *leaf);
    node *search(int key, node *leaf);
    void inorder_print(node *leaf);
    void postorder_print(node *leaf);
    void preorder_print(node *leaf);

    node *root;
};


btree::btree(){
    root = NULL;
}

btree::~btree(){
    destroy_tree();
}

void btree::destroy_tree(node *leaf){
    if(leaf != NULL)
```

```
    {
        destroy_tree(leaf->left);
        destroy_tree(leaf->right);
        delete leaf;
    }
}

void btree::insert(int key, node *leaf){

    if(key < leaf->value)
    {
        if(leaf->left != NULL)
        {
            insert(key, leaf->left);
        }
        else
        {
            leaf->left = new node;
            leaf->left->value = key;
            leaf->left->left = NULL;
            leaf->left->right = NULL;
        }
    }
    else if(key >= leaf->value)
    {
        if(leaf->right != NULL)
        {
            insert(key, leaf->right);
        }
        else
        {
            leaf->right = new node;
            leaf->right->value = key;
            leaf->right->right = NULL;
            leaf->right->left = NULL;
        }
    }
    cout<<"Element Inserted!\n";
}

void btree::insert(int key){
    if(root != NULL)
    {
        insert(key, root);
    }
    else
```

```
    {
        root = new node;
        root->value = key;
        root->left = NULL;
        root->right = NULL;
    }
}

node *btree::search(int key, node *leaf){
    if(leaf != NULL){
        if(key == leaf->value)
        {
            return leaf;
        }
        if(key < leaf->value)
        {
            return search(key, leaf->left);
        }
        else
        {
            return search(key, leaf->right);
        }
    }
    else
    {
        return NULL;
    }
}

node *btree::search(int key)
{
    return search(key, root);
}

void btree::destroy_tree()
{
    destroy_tree(root);
}

void btree::inorder_print()
{
    cout<<"InOrder = ";
    inorder_print(root);
    cout << "\n";
}
```

```
void btree::inorder_print(node *leaf)
{

    if(leaf != NULL){
        inorder_print(leaf->left);
        cout << leaf->value << ",";
        inorder_print(leaf->right);
    }
}

void btree::postorder_print()
{
    cout<<"PostOrder = ";
    postorder_print(root);
    cout << "\n";
}

void btree::postorder_print(node *leaf)
{

    if(leaf != NULL){
        inorder_print(leaf->left);
        inorder_print(leaf->right);
        cout << leaf->value << ",";
    }
}

void btree::preorder_print()
{
    cout<<"PreOrder = ";
    preorder_print(root);
    cout << "\n";
}

void btree::preorder_print(node *leaf)
{

    if(leaf != NULL)
    {
        cout << leaf->value << ",";
        inorder_print(leaf->left);
        inorder_print(leaf->right);
    }
}

int main()
```

```
{
    int choice, val;
    //btree tree;
    btree *tree = new btree();

    while(1)
    {
        cout<<"\n=========Opertaions on Binary Tree=============\n";
        cout<<"\n\t1.Insert element\n";
        cout<<"\t2.PreOrder print\n";
        cout<<"\t3.PostOrder Print\n";
        cout<<"\t4.Inorder Print\n";
        cout<<"\t5.Exit\n";
        cout<<" Enter you Choice: \n";
        cin>>choice;
        switch(choice)
        {
            case 1:
                cout<<"Enter Value to Be inserted: ";
                cin>> val;

                tree->insert(val);
                cout<<"Value " <<val <<" inserted.\n";
                break;
            case 2:
                tree->preorder_print();
                break;
            case 3:
                tree->postorder_print();
                break;
            case 4:
                tree->inorder_print();
                break;
            case 5:
                exit(0);
            default:
                cout<<"invalid Choice!\n";
        }
    }
    delete tree;

}
```

//DSAL_8_Selection_Sorting

```
#include <iostream>
using namespace std;

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    for (i = 0; i < n-1; i++)
    {
        min_idx = i;
        for (j = i+1; j < n; j++)
        {
            if (arr[j] < arr[min_idx])
            min_idx = j;
        }
        swap(&arr[min_idx], &arr[i]);
    }
}

void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main()
{
    int e;

    cout << "Enter the number of elements: ";
    cin >> e;
    int arr[e];
```

```
    cout << "Enter elements:" << '\n';
    for(int i = 0; i<e; i++)
    {
        cin >> arr[i];
    }

    int n = sizeof(arr)/sizeof(arr[0]);

    selectionSort(arr, n);

    cout << "Sorted array: \n";
    printArray(arr, n);

    return 0;
}
```

//DSAL_9_Bubble_Sorting

```
#include<iostream>
using namespace std;
void swapping(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
void display(int *array, int size)
{
    for(int i = 0; i<size; i++)
        cout << array[i] << " ";
    cout << endl;
}
void bubbleSort(int *array, int size)
{
    for(int i = 0; i<size; i++)
    {
        int swaps = 0;
        for(int j = 0; j<size-i-1; j++)
        {
            if(array[j] > array[j+1])
            {
                swapping(array[j], array[j+1]);
                swaps = 1;
            }
        }
        if(!swaps)
        break;
    }
}
int main()
{
    int n;

    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];

    cout << "Enter elements:" << '\n';
    for(int i = 0; i<n; i++)
    {
```

```
        cin >> arr[i];
    }
    cout << "Array before Sorting: ";
    display(arr, n);
    bubbleSort(arr, n);

    cout << "Array after Sorting: ";
    display(arr, n);

}
```

//DSAL_10_Insertion_sorting

```
#include<iostream>
using namespace std;
void display(int *array, int size)
{
    for(int i = 0; i<size; i++)
    {
        cout << array[i] << " ";
    }
    cout << '\n';
}

void insertionSort(int *array, int size)
{
    int key, j;
    for(int i = 1; i<size; i++)
    {
        key = array[i];
        j = i;

        while(j > 0 && array[j-1]>key)
        {
            array[j] = array[j-1];
            j--;
        }
        array[j] = key;
    }
}
int main()
{
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter elements:" << '\n';
    for(int i = 0; i<n; i++)
    {
        cin >> arr[i];
    }
    cout << "Array before Sorting: ";
    display(arr, n);
    insertionSort(arr, n);
    cout << "Array after Sorting: ";
    display(arr, n);
    }
```

//DSAL_11_Merge_sorting

```
#include<iostream>
using namespace std;
void swapping(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
void display(int *array, int size)
{
    for(int i = 0; i<size; i++)
    {
        cout << array[i] << " ";
    }
    cout << '\n';
}
void merge(int *array, int l, int m, int r)
{
    int i, j, k, nl, nr;
    nl = m-l+1; nr = r-m;
    int larr[nl], rarr[nr];
    for(i = 0; i<nl; i++)
    {
        larr[i] = array[l+i];
    }
    for(j = 0; j<nr; j++)
    {
        rarr[j] = array[m+1+j];
    }
    i = 0; j = 0; k = l;
    while(i < nl && j<nr)
    {
        if(larr[i] <= rarr[j])
        {
            array[k] = larr[i];
            i++;
        }
        else
        {
            array[k] = rarr[j];
            j++;
        }
        k++;
```

```
    }

    while(i<nl)
    {
        array[k] = larr[i];
        i++; k++;
    }
    while(j<nr)
    {
        array[k] = rarr[j];
        j++; k++;
    }
}
void mergeSort(int *array, int l, int r)
{
    int m;
    if(l < r)
    {
        int m = l+(r-l)/2;
        mergeSort(array, l, m);
        mergeSort(array, m+1, r);
        merge(array, l, m, r);
    }
}
int main()
{
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;

    int arr[n];

    cout << "Enter elements:" << '\n';

    for(int i = 0; i<n; i++)
    {
        cin >> arr[i];
    }

    cout << "Array before Sorting: ";
    display(arr, n);

    mergeSort(arr, 0, n-1);
    cout << "Array after Sorting: ";
    display(arr, n);
}
```

//DSAL_12_Heap_Sorting

```
#include <iostream>
using namespace std;

void heapify(int arr[], int n, int i)
{

    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
    {
        largest = left;
    }

    if (right < n && arr[right] > arr[largest])
    {
        largest = right;
    }

    if (largest != i)
    {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
    {
        heapify(arr, n, i);
    }

    for (int i = n - 1; i >= 0; i--)
    {
        swap(arr[0], arr[i]);

        heapify(arr, i, 0);
    }
}

void printArray(int arr[], int n)
{
```

```
    for (int i = 0; i < n; ++i)
    {
        cout << arr[i] << " ";
    }
    cout << "\n";
}

int main()
{
    int arr[] = {1, 12, 9, 5, 6, 10};

    int n = sizeof(arr) / sizeof(arr[0]);
    heapSort(arr, n);

    cout << "Sorted array is \n";
    printArray(arr, n);
}
```

//DSAL_13_Linear_Search

```
#include <iostream>
using namespace std;

int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
    {
        if (arr[i] == x)
        {
            return i;
        }
    }
    return -1;
}

int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int n = sizeof(arr) / sizeof(arr[0]);

    int result = search(arr, n, x);

    if(result == -1)
    {
        cout << "Element is not present in array";
    }
    else
    {
        cout << "Element is present at index " << result;
    }

    return 0;
}
```

//DSAL_14_Binary_Search

```
#include <iostream>
using namespace std;

int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l)
    {
        int mid = l + (r - l) / 2;

        if (arr[mid] == x)
        {
            return mid;
        }
        if (arr[mid] > x)
        {
            return binarySearch(arr, l, mid - 1, x);
        }
        return binarySearch(arr, mid + 1, r, x);
    }
    return -1;
}
int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };

    int x = 4;

    int n = sizeof(arr) / sizeof(arr[0]);

    int result = binarySearch(arr, 0, n - 1, x);

    if(result == -1)
    {
        cout << "Element is not present in array";
    }
    else
    {
        cout << "Element is present at index " << result;
    }
    return 0;
}
```