Data Structures and Algorithms

RECURSION 1

FACTORIAL :

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


int factorial(int n){
    if (n==0){
        return 1;
    }

    int smalloutput =factorial(n-1);
    return n*smalloutput ;

}

int main (){
    int n ;
    cin >>n;
    int output = factorial (n);
    cout << output <<endl;

}
```

FIBONACCI SERIES :

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

int fib(int n){

    if (n==0){
        return 0;
    }
    if (n==1){
        return 1;
    }
```

```
    int smallfib1=fib(n-1);
    int smallfib2=fib(n-2);

    return smallfib1+smallfib2;
}

int main (){
    cout <<fib(8);
}
```

NUMBER OF CHAR IN (OR LENGTH) A STRING :

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

int length (char s[]){

    if (s[0]=='\0'){
        return 0;
    }

    int smallstringlength = length (s+1);
    return 1+smallstringlength ;
}

int main (){

    char str[100];
    cin >> str ;
    int l=length(str);

    cout <<l;
}
```

STACKS USING ARRAYS

```
#include <climits>


class StackUsingArray {
```

```
int *data;
int nextIndex;
int capacity;

public :

StackUsingArray(int totalSize) {
	data = new int[totalSize];
	nextIndex = 0;
	capacity = totalSize;
}

// return the number of elements present in my stack
int size() {
	return nextIndex;
}

bool isEmpty() {
	/*
	if(nextIndex == 0) {
		return true;
	}
	else {
		return false;
	}
	*/

	return nextIndex == 0;
}

// insert element
void push(int element) {
	if(nextIndex == capacity) {
		cout << "Stack full " << endl;
		return;
	}
	data[nextIndex] = element;
	nextIndex++;
}

// delete element
int pop() {
	if(isEmpty()) {
		cout << "Stack is empty " << endl;
		return INT_MIN;
	}
	nextIndex--;
```

```
        return data[nextIndex];
    }

    int top() {
        if(isEmpty()) {
            cout << "Stack is empty " << endl;
            return INT_MIN;
        }
        return data[nextIndex - 1];
    }
```

STACK USING LINKED LIST

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

class Node {
    public :
        int data;
        Node *next;

        Node(int data) {
            this -> data = data;
            next = NULL;
        }
};


class Stack {
    Node *head;
    int size;          // number of elements prsent in stack

    public :

    Stack() {
        head = NULL;
        size = 0;
    }

    int getSize() {
        return size;
    }
```

```
    bool isEmpty() {
        return size == 0;
    }

    void push(int element) {
        Node *newNode = new Node(element);
        newNode -> next = head;
        head = newNode;
        size++;
    }

    int pop() {
        if(isEmpty()) {
            return 0;
        }
        int ans = head -> data;
        Node *temp = head;
        head = head -> next;
        delete temp;
        size--;
        return ans;
    }

    int top() {
        if(isEmpty()) {
            return 0;
        }
        return head -> data;

    }
};


int main() {
    Stack s;
    s.push(100);
    s.push(101);
    s.push(102);
    s.push(103);
    s.push(104);


    cout << s.top() << endl;

    cout << s.pop() << endl;
    cout << s.pop() << endl;
```

```
        cout << s.pop() << endl;

        cout << s.getSize() << endl;

        cout << s.isEmpty() << endl;

}
```

QUEUES USING ARRAYS

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



class QueueUsingArray {
        int *data;
        int nextIndex;
        int firstIndex;
        int size;
        int capacity;

        public :
        QueueUsingArray(int s) {
                data = new int[s];
                nextIndex = 0;
                firstIndex = -1;
                size = 0;
                capacity = s;
        }

        int getSize() {
                return size;
        }

        bool isEmpty() {
                return size == 0;
        }

        // insert element
        void enqueue(int element) {
                if(size == capacity) {
```

```
            cout << "Queue Full ! " << endl;
            return;
        }
        data[nextIndex] = element;
        nextIndex = (nextIndex + 1) % capacity ;
        if(firstIndex == -1) {
            firstIndex = 0;
        }
        size++;
    }

    int front() {
        if(isEmpty()) {
            cout << "Queue is empty ! " << endl;
            return 0;
        }
        return data[firstIndex];
    }

    int dequeue() {
        if(isEmpty()) {
            cout << "Queue is empty ! " << endl;
            return 0;
        }
        int ans = data[firstIndex];
        firstIndex = (firstIndex + 1) % capacity;
        size--;
        if(size == 0) {
            firstIndex = -1;
            nextIndex = 0;
        }
        return ans;
    }
};

int main() {
    QueueUsingArray q(5);

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.enqueue(50);
    q.enqueue(60);


    cout << q.front() << endl;
```

```
    cout << q.dequeue() << endl;
    cout << q.dequeue() << endl;
    cout << q.dequeue() << endl;

    cout << q.getSize() << endl;
    cout << q.isEmpty() << endl;
}
```