1)

a) FACTORIAL SOCKET PROG

```
//server.py

import socket
import math
s=socket.socket()
print('Socket created')
s.bind(('localhost',9999))
s.listen(1)
print('Waiting for connections')
while(True):
    c,addr = s.accept()
    print('connected with ',addr)
    num = c.recv(1024).decode('utf-8')
    n=int(num)
    f=math.factorial(n)
    print('factorial value: ',f)
    c.sendall(str(f).encode('utf-8'))
    c.close()
```

```
//client.py

import socket
c=socket.socket()
c.connect(('localhost',9999))
n=int(input("enter a number:"))
c.sendall(str(n).encode('utf-8'))
d=c.recv(1024).decode('utf-8')
print(int(d))
c.close()
```

b)   NETSTAT COMMAND

The `netstat` command is a network utility tool used to:

- Display active network connections on a computer.
- Show listening ports (where the computer is waiting for connections).
- Provide information about routing tables and interface statistics.

Basic Syntax:

```
netstat [options]
```

Common Options:

1. -a:Display all active connections.
2. -t: Display TCP connections.
3. -u: Display UDP connections.

4. -n: Show numerical addresses and port numbers.
5. -l: Display only listening sockets.
6. -r: Show the routing table.
7. -p: Display the process ID and program name for each socket.

Examples:

1. Display all active connections: `netstat -a`

2. Display TCP connections: `netstat -t`

3. Display listening sockets: `netstat -l`

c) DIJKSTRA SHORTEST PATH

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 9
int minDistance(int dist[], bool sptSet[])
{
        int min = INT_MAX, min_index;

        for (int v = 0; v < V; v++)
                if (sptSet[v] == false && dist[v] <= min)
                        min = dist[v], min_index = v;

        return min_index;
}
void printSolution(int dist[])
{
        printf("Vertex \t\t Distance from Source\n");
        for (int i = 0; i < V; i++)
                printf("%d \t\t\t\t %d\n", i, dist[i]);
}
void dijkstra(int graph[V][V], int src)
{
        int dist[V];
        bool sptSet[V];
        for (int i = 0; i < V; i++)
                dist[i] = INT_MAX, sptSet[i] = false;
        dist[src] = 0;
        for (int count = 0; count < V - 1; count++) {
                int u = minDistance(dist, sptSet);
                sptSet[u] = true;
                for (int v = 0; v < V; v++)
                        if (!sptSet[v] && graph[u][v]
                                && dist[u] != INT_MAX
                                && dist[u] + graph[u][v] < dist[v])
                                dist[v] = dist[u] + graph[u][v];
        }
        printSolution(dist);
}
int main()
{
        int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                                                    { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                                                    { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                                                    { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                                                    { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                                                    { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                                                    { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                                                    { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                                                    { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
        dijkstra(graph, 0);
        return 0;
}


/////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////
```

2)

a) SOCKET PRIME OR NOT

```
//server.py

import socket
s = socket.socket(socket.SOCK_DGRAM)
print('Socket Created ')
s.bind(('localhost',9999)) #9999 is port number
s.listen(3) #socket will listen for 3 connections
print('Waiting for connection')
def isprime(n):
    if n<=1:
        return False
    for i in range(2,n):
        if n%i == 0:
            return False
    return True
while True:
    c_socket,addr=s.accept() #for accepting the connection which returns client 
socket and address
    number=c_socket.recv(1024)
    data=int(number.decode('utf-8'))
    print(isprime(data))
    print('Client connected with',addr)
    c_socket.send(bytes('Welcome','utf-8')) #sending response to client
    c_socket.close()
```

```
//client.py

import socket
c = socket.socket()
c.connect(('localhost',9999))
number=int(input("Enter your number"))
data=str(number).encode('utf-8')
c.send(data)
print(c.recv(1024).decode())#1024 is buffer size
```

b) PING COMMAND

The ping command is a network utility tool used to test the reachability of a host (typically a computer or server) on an Internet Protocol (IP) network. It also measures the round-trip time it takes for a packet to travel from the source to the destination and back.

syntax : `ping [options] destination`

ex : `ping www.google.com`

c) LINK STATE ROUTING

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

#define MAX_ROUTERS 3

typedef struct {
    int cost;
    int state;  // 0: down, 1: up
} Link;

typedef struct {
    Link links[MAX_ROUTERS];
} Router;

typedef struct {
    int numRouters;
    Router routers[MAX_ROUTERS];
} NetworkState;

void initializeNetwork(NetworkState *network, int numRouters) {
    network->numRouters = numRouters;

    for (int i = 0; i < numRouters; i++) {
        for (int j = 0; j < numRouters; j++) {
            network->routers[i].links[j].cost = INT_MAX;
            network->routers[i].links[j].state = 0;  // All links initially down
        }
    }
}

void updateLinkState(NetworkState *network, int router1, int router2, int cost,
int state) {
    network->routers[router1].links[router2].cost = cost;
    network->routers[router1].links[router2].state = state;

    network->routers[router2].links[router1].cost = cost;
    network->routers[router2].links[router1].state = state;
}

void dijkstra(NetworkState *network, int source) {
    int distance[MAX_ROUTERS];

    for (int i = 0; i < network->numRouters; i++) {
        distance[i] = INT_MAX;
    }

    distance[source] = 0;

    for (int i = 0; i < network->numRouters - 1; i++) {
        int minDistance = INT_MAX, minIndex = -1;

        for (int j = 0; j < network->numRouters; j++) {
            if (distance[j] < minDistance) {
                minDistance = distance[j];
                minIndex = j;
            }
        }

        for (int j = 0; j < network->numRouters; j++) {
            int newDistance = distance[minIndex] +
network->routers[minIndex].links[j].cost;
            if (network->routers[minIndex].links[j].state && newDistance <
distance[j]) {
                distance[j] = newDistance;
            }
        }
    }

    printf("Shortest paths from router %d:\n", source);
    for (int i = 0; i < network->numRouters; i++) {
        printf("Router %d: %d\n", i, distance[i]);
    }
}

int main() {
    NetworkState network;
    initializeNetwork(&network, 3);

    updateLinkState(&network, 0, 1, 2, 1);
    updateLinkState(&network, 0, 2, 4, 1);
    updateLinkState(&network, 1, 2, 1, 1);

    dijkstra(&network, 0);

    return 0;
}
```

```
////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////
```

3)

a) LOWER UPPER CASE SOCKET

```
//server.py
import socket;
def start_server():
    server=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    server.bind(('localhost',3000))
    server.listen(5)
    while True:
        client_socket,addr=server.accept();
        data=client_socket.recv(1024).decode()
        if not data:
            break;
        result=None;
        data=str(data)
        if(data.islower()):
            result=data.upper();
        else:
            result=data.lower()
        client_socket.send(str(result).encode())
        client_socket.close()
start_server()

//client.py
import socket;
def start_client():
    client=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    s=input('Enter A string')
    client.connect(('localhost',3000))
    while s.strip().lower()!='bye':
        client.send(s.encode())
        data=client.recv(1024).decode()
        print(s,' ',data)
        s=input('Enter')
    client.close()
```

```
start_client()
```

b) The `ifconfig` command is used to display and configure network interfaces on Unix-like operating systems, such as Linux. It shows information like IP addresses and MAC addresses.

Example:
(type in command prompt)

```
ifconfig
```

Note: On modern systems, the `ip` command is preferred over `ifconfig` for more advanced functionality.

c) LEAKY BUCKET

```
#include <stdio.h>

int main() {
    int storage = 0;
    int no_of_queries = 4;
    int bucket_size = 10;
    int input_pkt_size = 4;
    int output_pkt_size = 1;

    for (int i = 0; i < no_of_queries; i++) {
        int size_left = bucket_size - storage;

        if (input_pkt_size <= size_left) {
            storage += input_pkt_size;
        } else {
            printf("Packet loss = %d\n", input_pkt_size);
        }

        printf("Buffer size = %d out of bucket size = %d\n", storage,
bucket_size);
        storage -= output_pkt_size;
    }

    return 0;
}
```

///

4) a and c both same as Q1

b)   NS LOOKUP COMMAND

The `nslookup` command is a network troubleshooting tool used for querying the Domain Name System (DNS) to obtain domain name or IP address information. This command is available on most operating systems, including Windows, Linux, and macOS.

basic syntax for using `nslookup`:

```
nslookup [domain or IP address] [DNS server]
```

Example, to look up the IP address of a domain:

```
nslookup example.com
```

If you want to query a specific DNS server, you can specify it:

```
nslookup example.com 8.8.8.8
```

In the example above, "8.8.8.8" is Google's public DNS server.

//

5)

a) BIT STUFFING SOCKET

//server.py

```
import socket

PORT = 8080
BUFFER_SIZE = 1024

def bit_stuffing(data):
    stuffed_data = ""
    count = 0
    for bit in data:
        stuffed_data += bit
        if bit == '1':
            count += 1
            if count == 5:
                stuffed_data += '0'
                count = 0
        else:
            count = 0
    return stuffed_data

def main():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind(('0.0.0.0', PORT))
    server_socket.listen(5)

    print(f"Server listening on port {PORT}...")

    client_socket, client_addr = server_socket.accept()
    print(f"Connection established with {client_addr}")

    data = client_socket.recv(BUFFER_SIZE).decode('utf-8')
    print(f"Received Data from client: {data}")

    stuffed_data = bit_stuffing(data)
    print(f"Stuffed Data: {stuffed_data}")

    client_socket.send(stuffed_data.encode('utf-8'))
    print("Stuffed Data sent to client")

    client_socket.close()
    server_socket.close()

if __name__ == "__main__":
    main()



//client.py


import socket

PORT = 8080
BUFFER_SIZE = 1024

def main():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('127.0.0.1', PORT))

    message = input("Enter the data to send: ")
    client_socket.send(message.encode('utf-8'))
    print(f"Data sent to server: {message}")

    received_data = client_socket.recv(BUFFER_SIZE).decode('utf-8')
    print(f"Received Stuffed Data from server: {received_data}")

    client_socket.close()

if __name__ == "__main__":
    main()
```

b)   PING COMMAND

The ping command is a network utility tool used to test the reachability of a host (typically a computer or server) on an Internet Protocol (IP) network. It also measures the round-trip time it takes for a packet to travel from the source to the destination and back.

```
syntax :  ping [options] destination
```

```
ex : ping www.google.com
```

c) TOKEN BUCKET

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // For sleep function

typedef struct {
    int tokens;
    int max_tokens;
    int refill_rate; // tokens per second
} TokenBucket;

void initialize_bucket(TokenBucket *bucket, int max_tokens, int refill_rate) {
    bucket->tokens = max_tokens;
    bucket->max_tokens = max_tokens;
    bucket->refill_rate = refill_rate;
}

void refill_tokens(TokenBucket *bucket) {
    bucket->tokens += bucket->refill_rate;
    if (bucket->tokens > bucket->max_tokens) {
        bucket->tokens = bucket->max_tokens;
    }
}

int consume_tokens(TokenBucket *bucket, int tokens) {
    if (bucket->tokens >= tokens) {
        bucket->tokens -= tokens;
        return 1; // Success
    }
    return 0; // Failure
}

int main() {
    TokenBucket bucket;
    initialize_bucket(&bucket, 10, 1); // Max 10 tokens, refill 1 token per second

    // Simulate token consumption
    for (int i = 0; i < 20; i++) {
        sleep(1); // Wait for 1 second
        refill_tokens(&bucket);
        if (consume_tokens(&bucket, 3)) {
            printf("Action allowed. Tokens left: %d\n", bucket.tokens);
        } else {
            printf("Action denied. Tokens left: %d\n", bucket.tokens);
        }
    }

    return 0;
}
```

```
/////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////
```

6)

a) SELECTIVE REPEAT SLIDING WINDOW SOCKET

```
//server.py

import socket

def server_program():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host = '127.0.0.1'
    port = 12346
    server_socket.bind((host, port))

    server_socket.listen(5)
    print("Server listening...")

    while True:
        client_socket, addr = server_socket.accept()
        print(f"Connection from {addr}")

        handle_client(client_socket)

def handle_client(client_socket):
    window_size = 3
    frame_count = 10

    for i in range(0, frame_count, window_size):
        window = range(i, min(i + window_size, frame_count))
        send_frames(client_socket, window)

    # Indicate the end of transmission
    client_socket.send("END".encode())

    client_socket.close()

def send_frames(client_socket, window):
    for frame in window:
        client_socket.send(str(frame).encode())
        print(f"Sent frame {frame}")

    # Simulate receiving acknowledgment
    ack = client_socket.recv(1024).decode()
    print(f"Received acknowledgment for frame {ack}")

if __name__ == "__main__":
    server_program()

//client.py

import socket

def client_program():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host = '127.0.0.1'
    port = 12346

    client_socket.connect((host, port))

    receive_frames(client_socket)

    client_socket.close()

def receive_frames(client_socket):
    while True:
        data = client_socket.recv(1024).decode()
        if not data:
            break
        print(f"Received frame {data}")

        ack = f"Ack for {data}"
        client_socket.send(ack.encode())
        print(ack)

if __name__ == "__main__":
    client_program()
```

b) IFCONFIG COMMAND

ifconfig is a command-line utility used in Unix and Unix-like operating systems to manage and configure network interfaces. Its primary functions include displaying information about active network interfaces, configuring interface parameters such as IP addresses and netmasks, and enabling or disabling network interfaces.

When invoked without specific arguments, ifconfig provides detailed information about all active network interfaces on the system, including their IP addresses, MAC (Media Access Control) addresses, and other relevant details. This makes it a handy tool for troubleshooting network-related issues and obtaining a quick overview of the system's network configuration.

c)   DISTANCE VECTOR ROUTING

```
#include<stdio.h>
struct node
{
        unsigned dist[20];
        unsigned from[20];
}rt[10];
int main()
{
        int dmat[20][20];
        int n,i,j,k,count=0;
        printf("\nEnter the number of nodes : ");
        scanf("%d",&n);
        printf("\nEnter the cost matrix :\n");
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                {
                        scanf("%d",&dmat[i][j]);
                        dmat[i][i]=0;
                        rt[i].dist[j]=dmat[i][j];
                        rt[i].from[j]=j;
                }
                do
                {
                        count=0;
                        for(i=0;i<n;i++)
                        for(j=0;j<n;j++)
                        for(k=0;k<n;k++)
                                if(rt[i].dist[j]>dmat[i][k]+rt[k].dist[j])
                                {
                                        rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
                                        rt[i].from[j]=k;
                                        count++;
                                }
                }while(count!=0);
                for(i=0;i<n;i++)
                {
                        printf("\n\nState value for router %d is \n",i+1);
                        for(j=0;j<n;j++)
                        {
                                printf("\t\nnode %d via %d 
Distance%d",j+1,rt[i].from[j]+1,rt[i].dist[j]);
                        }
                }
        printf("\n\n");
}
```

```
Example output
Enter the number of nodes : 4

Enter the cost matrix :
0 3 5 99
3 0 99 1
5 4 0 2
99 1 2 0


State value for router 1 is

node 1 via 1 Distance0
node 2 via 2 Distance3
node 3 via 3 Distance5
node 4 via 2 Distance4

State value for router 2 is

node 1 via 1 Distance3
node 2 via 2 Distance0
node 3 via 4 Distance3
node 4 via 4 Distance1

State value for router 3 is

node 1 via 1 Distance5
node 2 via 4 Distance3
node 3 via 3 Distance0
node 4 via 4 Distance2

State value for router 4 is

node 1 via 2 Distance4
node 2 via 2 Distance1
node 3 via 3 Distance2
node 4 via 4 Distance0
```

//

7)

a) socket programming to implement Go Back N sliding window protocol using connection-oriented iterative server.

```
//server.py
import socket

def server_program():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('127.0.0.1', 12345))
    server_socket.listen(1)

    print("Server listening on port 12345...")

    while True:
        connection, addr = server_socket.accept()
        print("Connection from", addr)

        receive_data(connection)

def receive_data(connection):
    window_size = 3
    expected_sequence = 0

    while True:
        data = connection.recv(1024).decode()

        if not data:
            break

        sequence_number = int(data.split(':')[0])
        message = data.split(':')[1]

        if sequence_number == expected_sequence:
            print(f"Received packet with sequence number {sequence_number}:
{message}")
            connection.send(f"Acknowledgment for packet
{sequence_number}".encode())
            expected_sequence += 1

        if sequence_number == window_size - 1:
            window_size += 1

    connection.close()

if __name__ == '__main__':
    server_program()

//client.py
import socket
import time

def client_program():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('127.0.0.1', 12345))

    window_size = 3
    sequence_number = 0

    while True:
        for i in range(window_size):
            message = f"{sequence_number}:{time.time()}"
            client_socket.send(message.encode())
            print(f"Sent packet with sequence number {sequence_number}")

            sequence_number += 1

        acknowledgment = client_socket.recv(1024).decode()
        print(acknowledgment)

    client_socket.close()

if __name__ == '__main__':
    client_program()
```

b) NETSTAT COMMAND

The `netstat` command is a network utility tool used to:

- Display active network connections on a computer.
- Show listening ports (where the computer is waiting for connections).
- Provide information about routing tables and interface statistics.

Basic Syntax:

```
netstat [options]
```

Common Options:

1. -a:Display all active connections.
2. -t: Display TCP connections.
3. -u: Display UDP connections.
4. -n: Show numerical addresses and port numbers.
5. -l: Display only listening sockets.
6. -r: Show the routing table.
7. -p: Display the process ID and program name for each socket.

Examples:

1. Display all active connections: `netstat -a`

2. Display TCP connections: `netstat -t`

3. Display listening sockets: `netstat -l`

c) LEAKY BUCKET

```
#include <stdio.h>

int main() {
    int storage = 0;
    int no_of_queries = 4;
    int bucket_size = 10;
    int input_pkt_size = 4;
    int output_pkt_size = 1;

    for (int i = 0; i < no_of_queries; i++) {
        int size_left = bucket_size - storage;

        if (input_pkt_size <= size_left) {
            storage += input_pkt_size;
        } else {
            printf("Packet loss = %d\n", input_pkt_size);
        }

        printf("Buffer size = %d out of bucket size = %d\n", storage,
bucket_size);
        storage -= output_pkt_size;
    }

    return 0;
}
```