SCHOOL OF TECHNOLOGY
PANDIT DEENDAYAL ENERGY UNIVERSITY
GANDHINAGAR, GUJARAT, INDIA

Computer Science & Engineering LAB File (2023-24) Design and Analysis of Algorithm Lab (20CP209P)

Student Name: Patel Shiv Vijaykumar

Enrollment No.: 22BCP317 Semester: 4

Division: 4

Group: G(8)

Instructor: Prof. Rajendra Choudhary

List of Practical

Exp. No.	Experiment Title	Date	Signature
1	Implement the following sorting in C programming language. a) Bubble sort b) Insertion sort c) Selection sort Now, measure the execution time and the number of steps required to execute each algorithm in best case, worst case, and average case.		
2	Implement the following sorting in C programming language. d) Merge sort e) Quick sort f) Radix sort Now, measure the execution time and the number of steps required to execute each algorithm in best case, worst case, and average case.		
3	Use singly linked lists to implement integers of unlimited size. Each node of the list should store one digit of the integer. You should implement addition, subtraction, multiplication, and exponentiation operations. Limit exponents to be positive integers. What is the asymptotic running time for each of your operations, expressed in terms of the number of digits for the two operands of each function?		
4 (I) 4 (II)	Implement a city database using unordered lists. Each database record contains the name of the city (a string of arbitrary length) and the coordinates of the city expressed as integer x and y coordinates. Your program should allow following functionalities: a) Insert a record, b) Delete a record by name or coordinate, c) Search a record by name or coordinate. d) Pint all records within a given distance of a specified point. Implement the database using an array-based list implementation, and then a linked list implementation. Perform following analysis: a) Collect running time statistics for each operation in both implementations. b) What are your conclusions about the relative advantages and disadvantages of the two implementations? c) Would storing records on the list in alphabetical order by city name speed any of the operations? d) Would keeping the list in alphabetical order slow any of the operations?		

5 **[Greedy Approach]**

Implement interval scheduling algorithm. Given n events with their starting and ending times, find a schedule that includes as many events as possible. It is not possible to select an event partially. For example, consider the following example:

Event	**Starting time**	**Ending time**
A	1	3
B	2	5
C	3	9
D	6	8

Here, maximum number of events that can be scheduled is 2. We can schedule B and D together.

6 **[Divide and Conquer]**

Implement both a standard $O(n^3)$ matrix multiplication algorithm and Strassen's matrix multiplication algorithm. Using empirical testing, try and estimate the constant factors for the runtime equations of the two algorithms. How big must n be before Strassen's algorithm becomes more efficient than the standard algorithm?

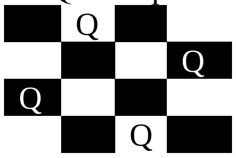
7 **[Dynamic Programming]**

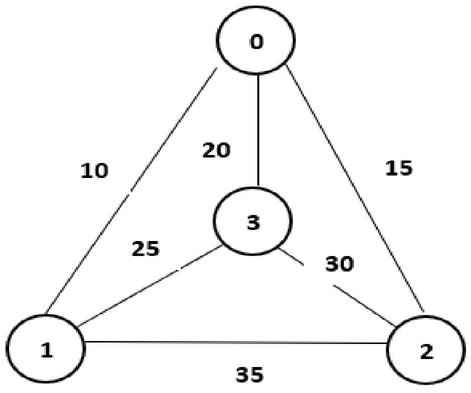
mplement the Floyd Warshall Algorithm for All Pair Shortest Path Problem. You are given a weighted diagraph $G = (V, E)$, with arbitrary edge weights or costs c_{vw} between any node v and node w. Find the cheapest path from every node to every other node. Edges may have negative weights. Consider the following test case to check your algorithm:

$\boldsymbol{v}$	$\boldsymbol{w}$	c_{vw}
0	1	-1
0	2	4
1	2	3
1	3	2
1	4	2
3	2	5
3	1	1
4	3	-3

8 **[Backtracking]**

Solve the n queens' problem using backtracking. Here, the task is to place n chess queens on an n x n board so that no two queens attack each other. For example, following is a solution for the 4 Queen' problem.



9	**[Branch and Bound]** Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point. Solve this problem using branch and bound technique. For example, consider the following graph:  A Travelling Salesman Problem (TSP) tour in the graph is $0-1-3-2-0$. The cost of the tour is $10+25+30+15=80$.		
10	To design and solve given problems using different algorithmic approaches and analyze their complexity.		
(I)	Your friends are starting a security company that needs to obtain licenses for n different pieces of cryptographic software. Due to regulations, they can only obtain these licenses at the rate of at most one per month. Each license is currently selling for a price of \$100. However, they are all becoming more expensive according to exponential growth curves: in particular, the cost of license j increases by a factor of $r_j > 1$ each month, where r_j is a given parameter. This means that if license j is purchased t months from now, it will cost $100r_jt$. We will assume that all the price growth rates are distinct; that is, $r_i \neq r_j$ for licenses $i \neq j$ (even though they start at the sameprice of \$100). The question is: Given that the company can only buy at most one license a month, in which order should it buy the licenses so that the total amount of money it spends is as small as possible? Give an algorithm that takes the n rates of price growth $r_1, r_2, \ldots, r_n$, and computes an order in which to buy the licenses so that the total amount ofmoney spent is minimized. The running time of your algorithm should be polynomial in n.		
(II)	Suppose you are given an array A with n entries, with each entry holding a distinct number. You are told that the sequence of values $A[1], A[2], \ldots, A[n]$ is unimodal. That is, for some index p between 1 and n, the values in the array entries increase up to position p in A and then decrease the remainder of the way until position n. (So if you were to draw a plot with the array position j on the x-axis and the value of the entry $A[j]$ on the y-axis, the plotted points would rise until x-value p, where they'd achieve their maximum value, and then fall from there on). You'd like to find the "peak entry" p without having to read the entire array - in fact, by reading as few entries of A as possible. Show how to find the entry p by reading at most		

	$O(logn)$ entries of A.		

Instruction's

i. Make all the programs using C language
ii. You are allowed to use only gcc compiler and command prompt for running the programming
iii. With each program you have to print your name and enrollment number.
iv. In each lab after making the programs, paste the code (in text format) with the output (snapshot of the output) in this file.
v. You have to submit only soft-copy of the lab manual.

1	Implement the following sorting in C programming language. a) Bubble sort b) Insertion sort c) Selection sort Now, measure the execution time and the number of steps required to execute each algorithm in best case, worst case, and average case.

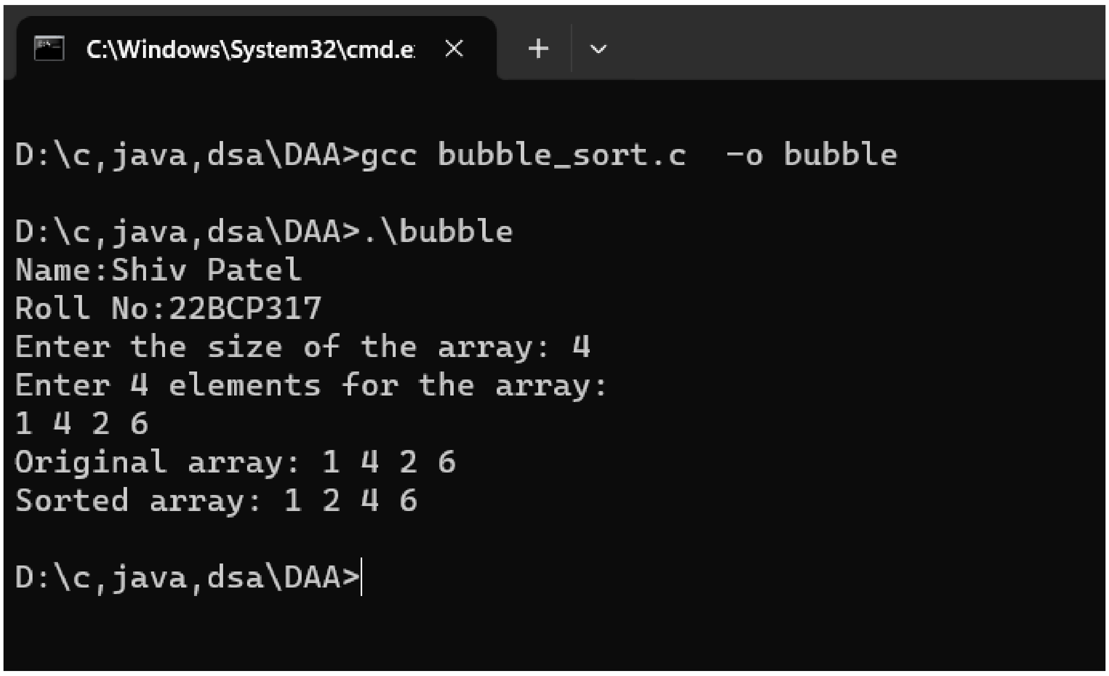
CODE:

A) BUBBLE SORT:

SOURCE CODE:

```
#include<stdio.h>

void printarray(int *a, int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");
}

void bubblesort(int *a, int n) {
    int temp;

    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1 - i; j++) {
            if (a[j] > a[j + 1]) {
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
}

int main() {

    printf("Name: Shiv Patel\n");
    printf("Roll No:22BCP317\n");

    int n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int a[n];
    printf("Enter %d elements for the array:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    printf("Original array: ");
    printarray(a, n);

    bubblesort(a, n);

    printf("Sorted array: ");
    printarray(a, n);

    return 0;
}
```

OUTPUT:



```
D:\c,java,dsa\DAA>gcc bubble_sort.c  -o bubble

D:\c,java,dsa\DAA>.\bubble
Name:Shiv Patel
Roll No:22BCP317
Enter the size of the array: 4
Enter 4 elements for the array:
1 4 2 6
Original array: 1 4 2 6
Sorted array: 1 2 4 6

D:\c,java,dsa\DAA>
```

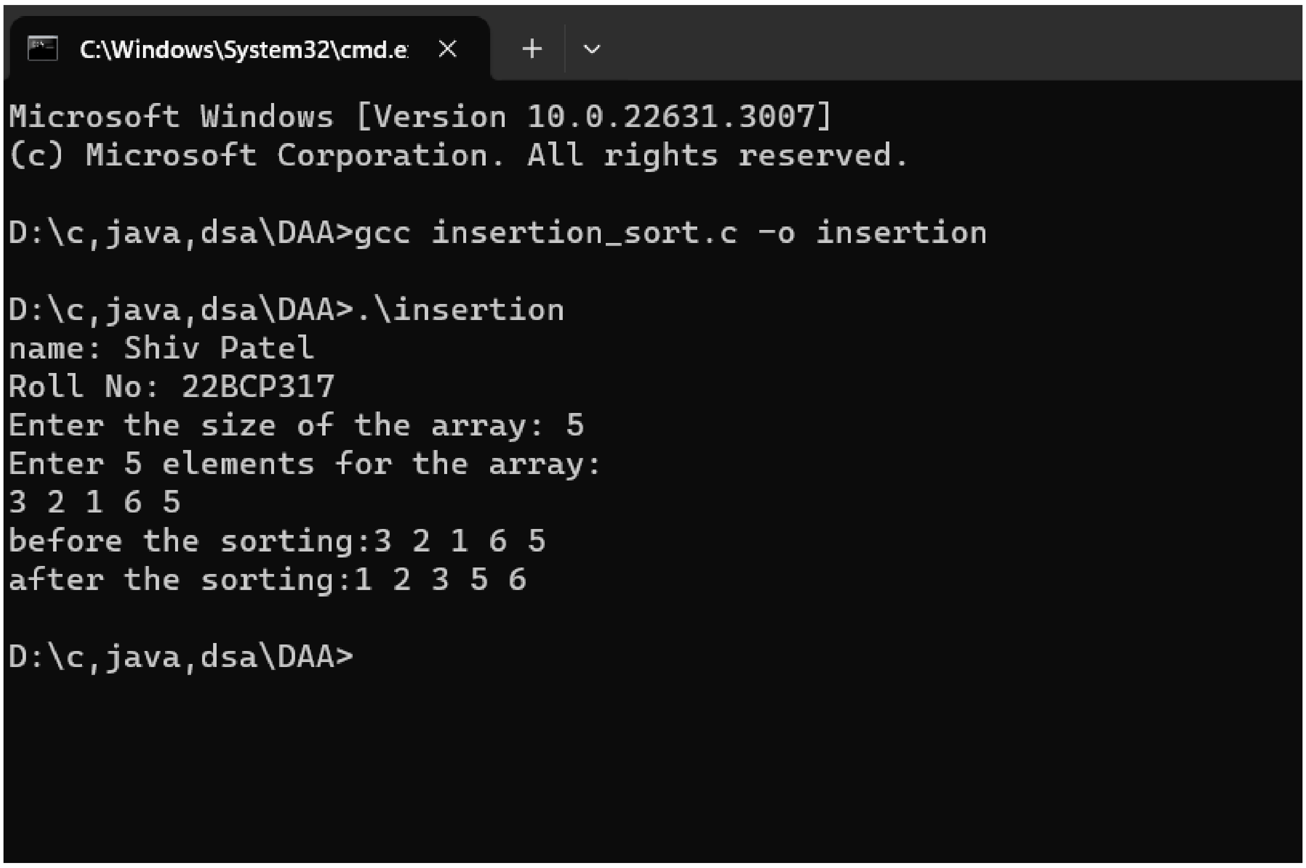
B) Insertion sort:

SOURCE CODE:

```
#include<stdio.h>

void printarray(int *a ,int n){
    for(int i=0 ; i<n ; i++){
      printf("%d ",a[i]);
    }
    printf("\n");
}

void insertionsort(int *a,int n){
    int key , j ;
    for ( int i=1 ; i<n ; i++){
        key = a[i];
        j = i-1;

        while(j>=0 && a[j]>key){
            a[j+1]=a[j];
            j--;
        }
        a[j+1]=key;
    }
}

int main()

{

    int n;

    printf("name: Shiv Patel\n");
    printf("Roll No: 22BCP317\n");
    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int a[n];

    printf("Enter %d elements for the array:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    printf("before the sorting:");
    printarray(a,n);
    insertionsort(a,n);
```

```
 printf("after the sorting:");
 printarray(a,n);
 return 0;
}
```

OUTPUT:



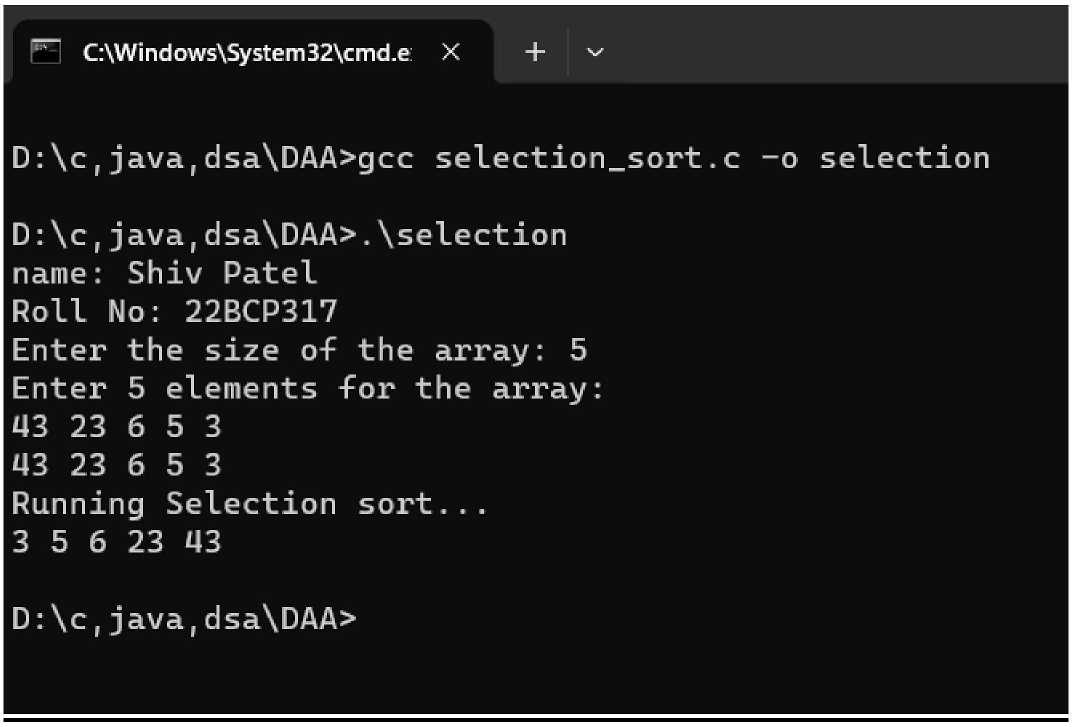
```
Microsoft Windows [Version 10.0.22631.3007]
(c) Microsoft Corporation. All rights reserved.

D:\c,java,dsa\DAA>gcc insertion_sort.c -o insertion

D:\c,java,dsa\DAA>.\insertion
name: Shiv Patel
Roll No: 22BCP317
Enter the size of the array: 5
Enter 5 elements for the array:
3 2 1 6 5
before the sorting:3 2 1 6 5
after the sorting:1 2 3 5 6

D:\c,java,dsa\DAA>
```

C) Selection sort:

SOURCE CODE:

```
#include<stdio.h>

void printArray(int *a, int n){
    for (int i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    printf("\n");
}

void selectionSort(int *a, int n){
    int indexOfMin, temp;
    printf("Running Selection sort...\n");
    for (int i = 0; i < n-1; i++)
    {
        indexOfMin = i;
        for (int j = i+1; j < n; j++)
        {
            if(a[j] < a[indexOfMin]){
                indexOfMin = j;
            }
        }
]
        temp = a[i];
        a[i] = a[indexOfMin];
        a[indexOfMin] = temp;
    }
}

int main(){

    int n;

    printf("name: Shiv Patel\n");
    printf("Roll No: 22BCP317\n");
    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int a[n];
    printf("Enter %d elements for the array:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }
    printArray(a, n);
    selectionSort(a, n);
    printArray(a, n);

    return 0;
}
```

OUTPUT:



```
D:\c,java,dsa\DAA>gcc selection_sort.c -o selection

D:\c,java,dsa\DAA>.\selection
name: Shiv Patel
Roll No: 22BCP317
Enter the size of the array: 5
Enter 5 elements for the array:
43 23 6 5 3
43 23 6 5 3
Running Selection sort...
3 5 6 23 43

D:\c,java,dsa\DAA>
```

2	Implement the following sorting in C programming language. a) Merge sort b) Quick sort c) Radix sort Now, measure the execution time and the number of steps required to execute each algorithm in best case, worst case, and average case.

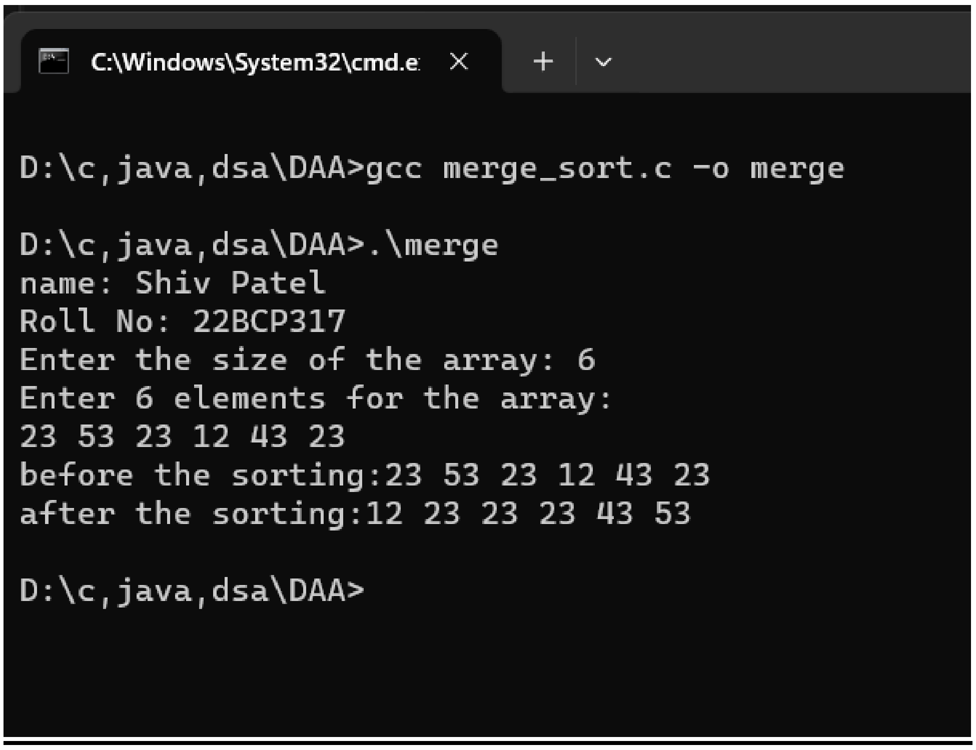
CODE:

a) Merge sort:

SOURCE CODE:

```
#include <stdio.h>

void printarray(int *a, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    printf("\n");
}

void merge(int a[], int mid, int low, int high)
{
    int i, j, k, B[100];
    i = low;
    j = mid + 1;
    k = low;

    while (i <= mid && j <= high)
    {
        if (a[i] < a[j])
        {
            B[k] = a[i];
            i++;
            k++;
        }
        else
        {
            B[k] = a[j];
            j++;
            k++;
        }
    }
    while (i <= mid)
    {
        B[k] = a[i];
        k++;
        i++;
    }

    while (j <= high)
    {
        B[k] = a[j];
        k++;
        j++;
    }

    for (int i = low; i <= high; i++)
    {
        a[i] = B[i];
    }

}

void mergesort(int a[], int low, int high){
    int mid;
    if(low<high) {
        mid = (low + high) /2;
        mergesort(a, low, mid);
        mergesort(a, mid+1, high);
        merge(a, mid, low, high);
    }
}

int main()
{
    int n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int a[n];

    printf("Enter %d elements for the array:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }
    printarray(a, n);
    mergesort(a, 0, n);
    printarray(a, n);
    return 0;
   }
```

OUTPUT:



```
D:\c,java,dsa\DAA>gcc merge_sort.c -o merge

D:\c,java,dsa\DAA>.\merge
name: Shiv Patel
Roll No: 22BCP317
Enter the size of the array: 6
Enter 6 elements for the array:
23 53 23 12 43 23
before the sorting:23 53 23 12 43 23
after the sorting:12 23 23 23 43 53

D:\c,java,dsa\DAA>
```

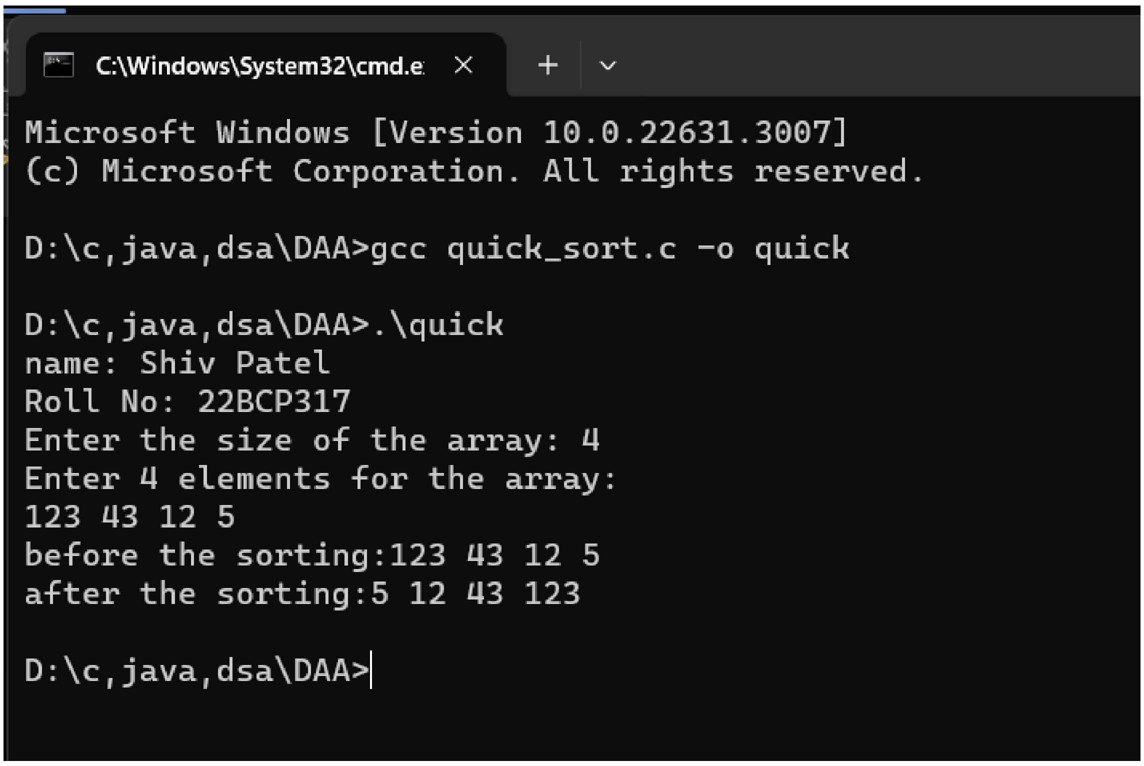
b) <u>QUICK SORT:</u>

SOURCE CODE:

```c
#include <stdio.h>

void printarray(int *A, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", A[i]);
    }
    printf("\n");
}

int partition(int A[], int low, int high)
{
    int pivot = A[low];
    int i = low + 1;
    int j = high;
    int temp;

    do
    {
        while (A[i] <= pivot)
        {
            i++;
        }

        while (A[j] > pivot)
        {
            j--;
        }


        if (i < j)
        {
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
        }
    } while (i < j);

    temp = A[low];
    A[low] = A[j];
    A[j] = temp;
    return j;
}
```

```
void quicksort(int A[], int low, int high)
{
    int partitionindex;

    if (low < high)
    {
        partitionindex = partition(A, low, high);
        quicksort(A, low, partitionindex - 1);
        quicksort(A, partitionindex + 1, high);
    }
}

int main()
{
   int n;
    printf("name: Shiv Patel\n");
    printf("Roll No: 22BCP317\n");

    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int A[n];

    printf("Enter %d elements for the array:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &A[i]);
    }

    printf("before the sorting:");
    printarray(A, n);
    quicksort(A, 0, n - 1);
    printf("after the sorting:");
    printarray(A, n);
    return 0;
   }
```

Output:



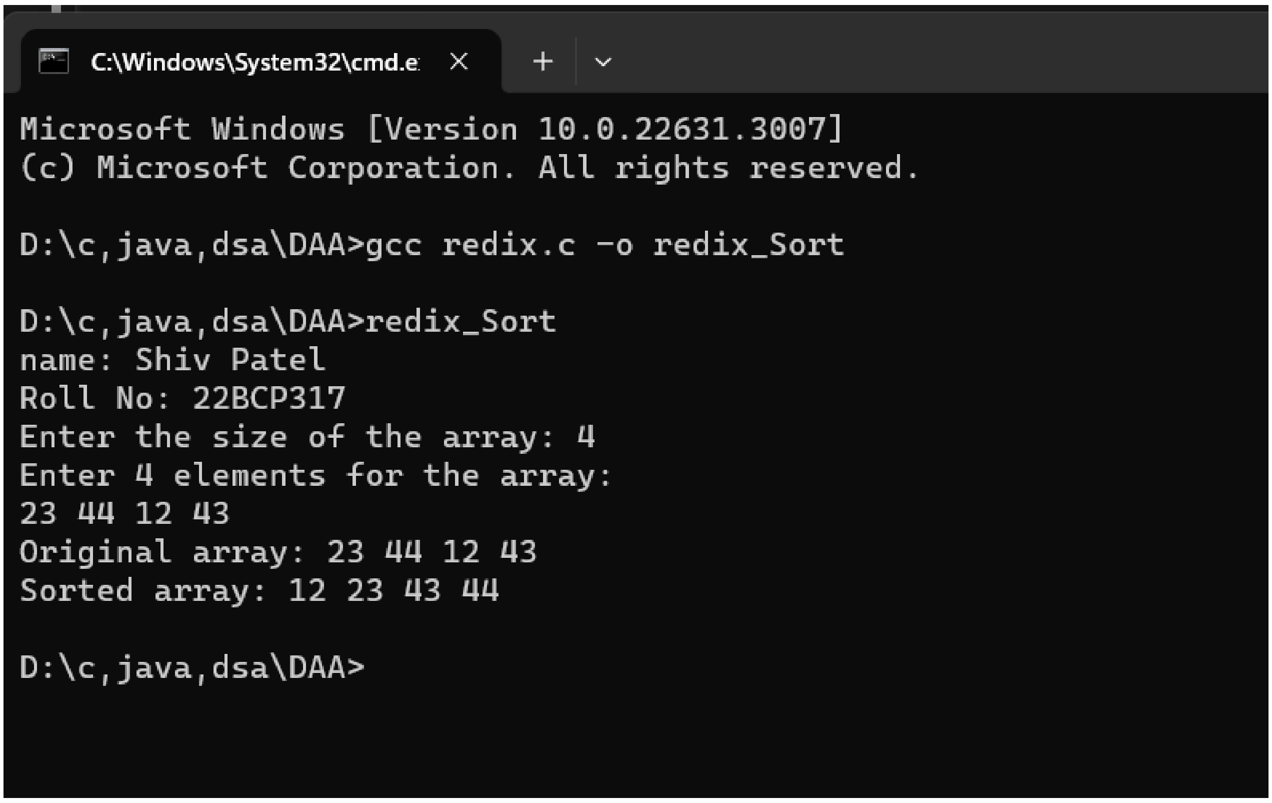
```
Microsoft Windows [Version 10.0.22631.3007]
(c) Microsoft Corporation. All rights reserved.

D:\c,java,dsa\DAA>gcc quick_sort.c -o quick

D:\c,java,dsa\DAA>.\quick
name: Shiv Patel
Roll No: 22BCP317
Enter the size of the array: 4
Enter 4 elements for the array:
123 43 12 5
before the sorting:123 43 12 5
after the sorting:5 12 43 123

D:\c,java,dsa\DAA>
```

C) radixsort:

Source code:

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

void printarray(int A[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", A[i]);
    }
    printf("\n");
}

int findmax(int A[], int n) {
    int max = A[0];
    for (int i = 1; i < n; i++) {
        if (A[i] > max) {
            max = A[i];
        }
    }
    return max;
}

void countingsort(int A[], int n, int exp) {
    const int base = 10;
    int output[n];
    int count[base];


    for (int i = 0; i < base; i++) {
        count[i] = 0;
    }

    for (int i = 0; i < n; i++) {
        count[(A[i] / exp) % base]++;
    }

    for (int i = 1; i < base; i++) {
        count[i] += count[i - 1];
    }

    for (int i = n - 1; i >= 0; i--) {
        output[count[(A[i] / exp) % base] - 1] = A[i];
        count[(A[i] / exp) % base]--;
    }
    for (int i = 0; i < n; i++) {
        A[i] = output[i];
    }
}

void radixsort(int A[], int n) {

    int max = findmax(A, n);

    for (int exp = 1; max / exp > 0; exp *= 10) {
        countingsort(A, n, exp);
    }
}

int main() {
    int n;
    printf("name: Shiv Patel\n");
    printf("Roll No: 22BCP317\n");

    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int A[n];

    printf("Enter %d elements for the array:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &A[i]);
    }

    printf("Original array: ");
    printarray(A, n);

    radixsort(A, n);

    printf("Sorted array: ");
    printarray(A, n);

    return 0;
}
```

Output:


C:\Windows\System32\cmd.e

```
Microsoft Windows [Version 10.0.22631.3007]
(c) Microsoft Corporation. All rights reserved.

D:\c,java,dsa\DAA>gcc redix.c -o redix_Sort

D:\c,java,dsa\DAA>redix_Sort
name: Shiv Patel
Roll No: 22BCP317
Enter the size of the array: 4
Enter 4 elements for the array:
23 44 12 43
Original array: 23 44 12 43
Sorted array: 12 23 43 44

D:\c,java,dsa\DAA>
```

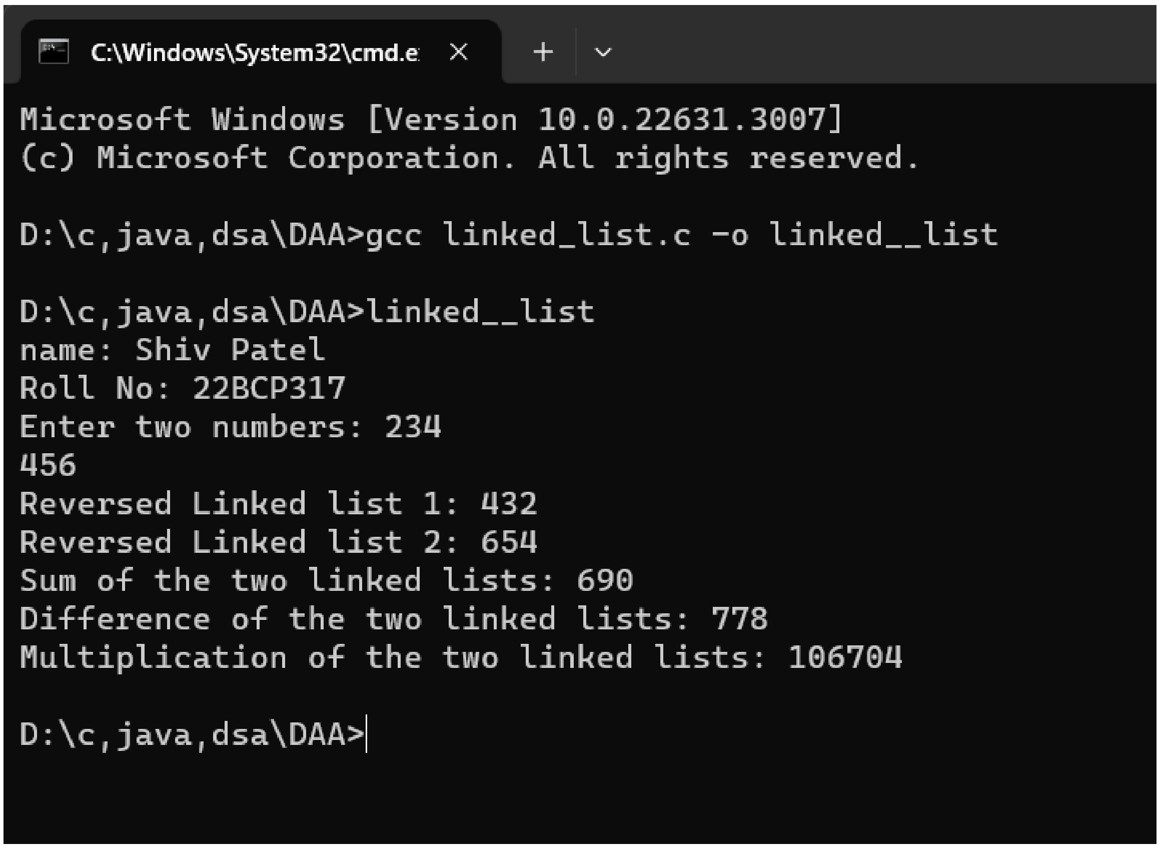
3	Use singly linked lists to implement integers of unlimited size. Each node of the list should store one digit of the integer. You should implement addition, subtraction, multiplication, and exponentiation operations. Limit exponents to be positive integers. What is the asymptotic running time for each of your operations, expressed in terms of the number of digits for the two operands of each function?

Source code:

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

struct node
{
   int data;
   struct node *next;
};

struct node *createnode(int data)
{
   struct node *newnode = (struct node *)malloc(sizeof(struct node));
   newnode->data = data;
   newnode->next = NULL;
   return newnode;
}

void insertend(struct node *headRef, int data)
{
   struct node *newnode = createnode(data);
   if (*headRef == NULL)
   {
      *headRef = newnode;
      return;

   struct node *temp = *headRef;
   while (temp->next != NULL)
   {
      temp = temp->next;
   }
   temp->next = newnode;
}

struct node *addlinkedlists(struct node *head1, struct node *head2)
{
   struct node *result = NULL;
   struct node *tail = NULL;
   int carry = 0, sum;

   while (head1 != NULL || head2 != NULL || carry != 0)
   {
      sum = carry;
      if (head1 != NULL)
      {
         sum += head1->data;
```

```
            head1 = head1->next;
        }
        if (head2 != NULL)
        {
            sum += head2->data;
            head2 = head2->next;
        }

        carry = sum / 10;
        sum = sum % 10;

        struct node *newnode = createnode(sum);

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

    return result;
}

struct node *subtractlinkedlists(struct node *head1, struct node *head2)
{

    struct node *result = NULL;
    struct node *tail = NULL;
    int borrow = 0, diff;

    while (head1 != NULL || head2 != NULL)
    {
        int num1 = (head1 != NULL) ? head1->data : 0;
        int num2 = (head2 != NULL) ? head2->data : 0;

        diff = num1 - num2 - borrow;

        if (diff < 0)
        {
            diff = diff + 10;
            borrow = 1;
        }
        else
        {
            borrow = 0;
        }

        struct node *newnode = createnode(diff);

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

        if (head1 != NULL)
            head1 = head1->next;
        if (head2 != NULL)
            head2 = head2->next;
    }

    return result;
}

struct node *multiplylinkedlists (struct node *head1, struct node *head2)
{
    struct node *result = NULL;

    if (head1 == NULL || head2 == NULL)
        return result;

    int position = 0;

    while (head2 != NULL)
    {

        struct node *tempresult = NULL;
        struct node *temptail = NULL;

        int carry = 0;

        struct node *temphead1 = head1;
        while (temphead1 != NULL)
        {

            int product = (temphead1->data) * (head2->data) + carry;

            carry = product / 10;
            product %= 10;

            struct node *newnode = createnode(product);

            if (tempresult == NULL)
            {
                tempresult = newnode;
                temptail = tempresult;
            }
            else
            {
                temptail->next = newnode;
                temptail = temptail->next;
            }

            temphead1 = temphead1->next;
        }
        if (carry > 0)
        {
            temptail->next = createnode(carry);
        }

        for (int i = 0; i < position; i++)
        {
            struct node *zeronode = createnode(0);
            zeronode->next = tempresult;
            tempresult = zeronode;
        }

        result = addlinkedlists(result, tempresult);

        head2 = head2->next;

        position++;
    }

    return result;
}

void printlist(struct node *head)
{
    while (head != NULL)
    {
        printf("%d", head->data);
        head = head->next;
    }
    printf("\n");
}

void printreverse(struct node *head)
{
    if (head == NULL)
        return;

    printreverse(head->next);


    printf("%d", head->data);
}
int main()
{
    int num1, num2;

    printf("name: Shiv Patel\n");
    printf("Roll No: 22BCP317\n");

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    struct node *head1 = NULL;
    struct node *head2 = NULL;

    while (num1 > 0)
    {
        insertend(&head1, num1 % 10);
        num1 = num1 / 10;
    }

    while (num2 > 0)
    {
        insertend(&head2, num2 % 10);
        num2 = num2 / 10;
    }

    printf("Reversed Linked list 1: ");
    printlist(head1);

    printf("Reversed Linked list 2: ");
    printlist(head2);

    struct node *sum = addlinkedlists(head1, head2);
    printf("Sum of the two linked lists: ");
    printreverse(sum);
    printf("\n");

    struct node *diff = subtractlinkedlists(head1, head2);
    printf("Difference of the two linked lists: ");
    printreverse(diff);
    printf("\n");

    struct node *mul = multiplylinkedlists(head1, head2);
    printf("Multiplication of the two linked lists: ");
    printreverse(mul);
    printf("\n");

    return 0;
}
```

Output:



```
Microsoft Windows [Version 10.0.22631.3007]
(c) Microsoft Corporation. All rights reserved.

D:\c,java,dsa\DAA>gcc linked_list.c -o linked__list

D:\c,java,dsa\DAA>linked__list
name: Shiv Patel
Roll No: 22BCP317
Enter two numbers: 234
456
Reversed Linked list 1: 432
Reversed Linked list 2: 654
Sum of the two linked lists: 690
Difference of the two linked lists: 778
Multiplication of the two linked lists: 106704

D:\c,java,dsa\DAA>
```

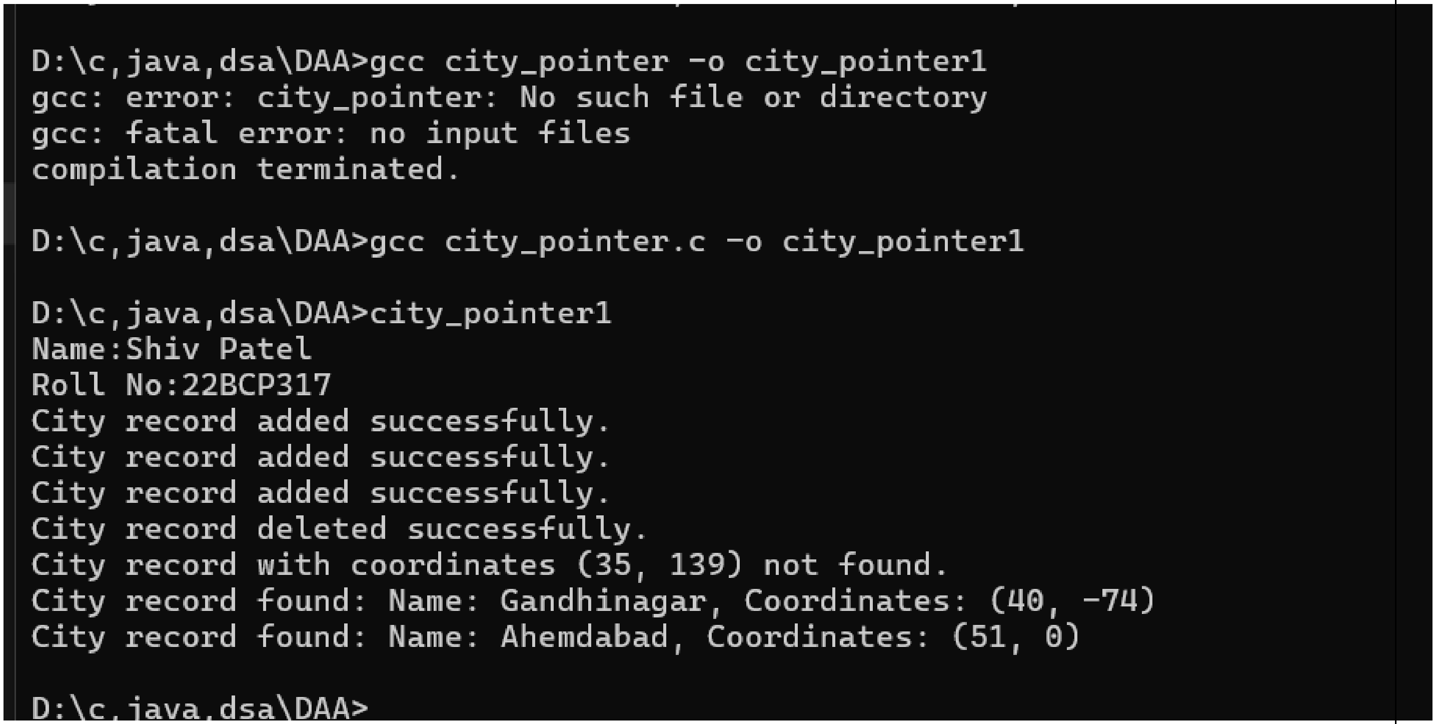
4 (I)	Implement a city database using unordered lists. Each database record contains the name of the city (a string of arbitrary length) and the coordinates of the city expressed as integer x and y coordinates. Your program should allow following functionalities: e) Insert a record, f) Delete a record by name or coordinate, g) Search a record by name or coordinate. h) Pint all records within a given distance of a specified point.

USING LINKED-LIST:

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

struct city
{
   char name[100];
   int x;
   int y;
};

struct city_database
{
   struct city cities[100];
   int num_cities;
};

void insert_record(struct city_database *db, const char *name, int x, int y)
{
   if (db->num_cities < 100)
   {
      strcpy(db->cities[db->num_cities].name, name);
      db->cities[db->num_cities].x = x;
      db->cities[db->num_cities].y = y;
      db->num_cities++;
      printf("city record added successfully.\n");
   }
   else
   {
      printf("city database is full. Cannot add record.\n");
   }
}

void delete_record_by_name(struct city_database *db, const char *name)
{
   int found = 0;
   for (int i = 0; i < db->num_cities; i++)
   {
      if (strcmp(db->cities[i].name, name) == 0)
      {
         for (int j = i; j < db->num_cities - 1; j++)
         {
            db->cities[j] = db->cities[j + 1];
         }
         db->num_cities--;
```

```
            found = 1;
            printf("city record deleted successfully.\n");
            break;
        }
    }
    if (!found)
    {
        printf("city record with name '%s' not found.\n", name);
    }
}

void delete_record_by_coordinates(struct city_database *db, int x, int y)
{
    int found = 0;
    for (int i = 0; i < db->num_cities; i++)
    {
        if (db->cities[i].x == x && db->cities[i].y == y)
        {
            for (int j = i; j < db->num_cities - 1; j++)
            {
                db->cities[j] = db->cities[j + 1];
            }
            db->num_cities--;
            found = 1;
            printf("city record deleted successfully.\n");
            break;
        }
    }
    if (!found)
    {
        printf("city record with coordinates (%d, %d) not found.\n", x, y);
    }
}

void search_record_by_name(const struct city_database *db, const char *name)
{
    int found = 0;
    for (int i = 0; i < db->num_cities; i++)
    {
        if (strcmp(db->cities[i].name, name) == 0)
        {
            printf("city record found: Name: %s, Coordinates: (%d, %d)\n", db->cities[i].name,
                    db->cities[i].x, db->cities[i].y);
            found = 1;
            break;
        }
    }
    if (!found)
    {
        printf("city record with name '%s' not found.\n", name);
    }
}

void search_record_by_coordinates(const struct city_database *db, int x, int y)
{
    int found = 0;
    for (int i = 0; i < db->num_cities; i++)
    {
        if (db->cities[i].x == x && db->cities[i].y == y)
        {
            printf("city record found: Name: %s, Coordinates: (%d, %d)\n", db->cities[i].name, db->cities[i].x,
        db->cities[i].y);
            found = 1;
            break;
        }
    }
    if (!found)
    {
        printf("city record with coordinates (%d, %d) not found.\n", x, y);
    }
}

int main()
{

    printf("Name:Shiv Patel\n");
    printf("Roll No:22BCP317\n");

    struct city_database db;
    db.num_cities = 0;

    insert_record(&db, "Gandhinagar", 40, -74);
    insert_record(&db, "Ahemdabad", 51, 0);
    insert_record(&db, "Mehsana", 35, 139);

    delete_record_by_name(&db, "Mehsana");
    delete_record_by_coordinates(&db, 35, 139);

    search_record_by_name(&db, "Gandhinagar");
    search_record_by_coordinates(&db, 51, 0);

    return 0;
}
```

OUTPUT:



```
D:\c,java,dsa\DAA>gcc city_pointer -o city_pointer1
gcc: error: city_pointer: No such file or directory
gcc: fatal error: no input files
compilation terminated.

D:\c,java,dsa\DAA>gcc city_pointer.c -o city_pointer1

D:\c,java,dsa\DAA>city_pointer1
Name:Shiv Patel
Roll No:22BCP317
City record added successfully.
City record added successfully.
City record added successfully.
City record deleted successfully.
City record with coordinates (35, 139) not found.
City record found: Name: Gandhinagar, Coordinates: (40, -74)
City record found: Name: Ahemdabad, Coordinates: (51, 0)

D:\c,java,dsa\DAA>
```

Using array:

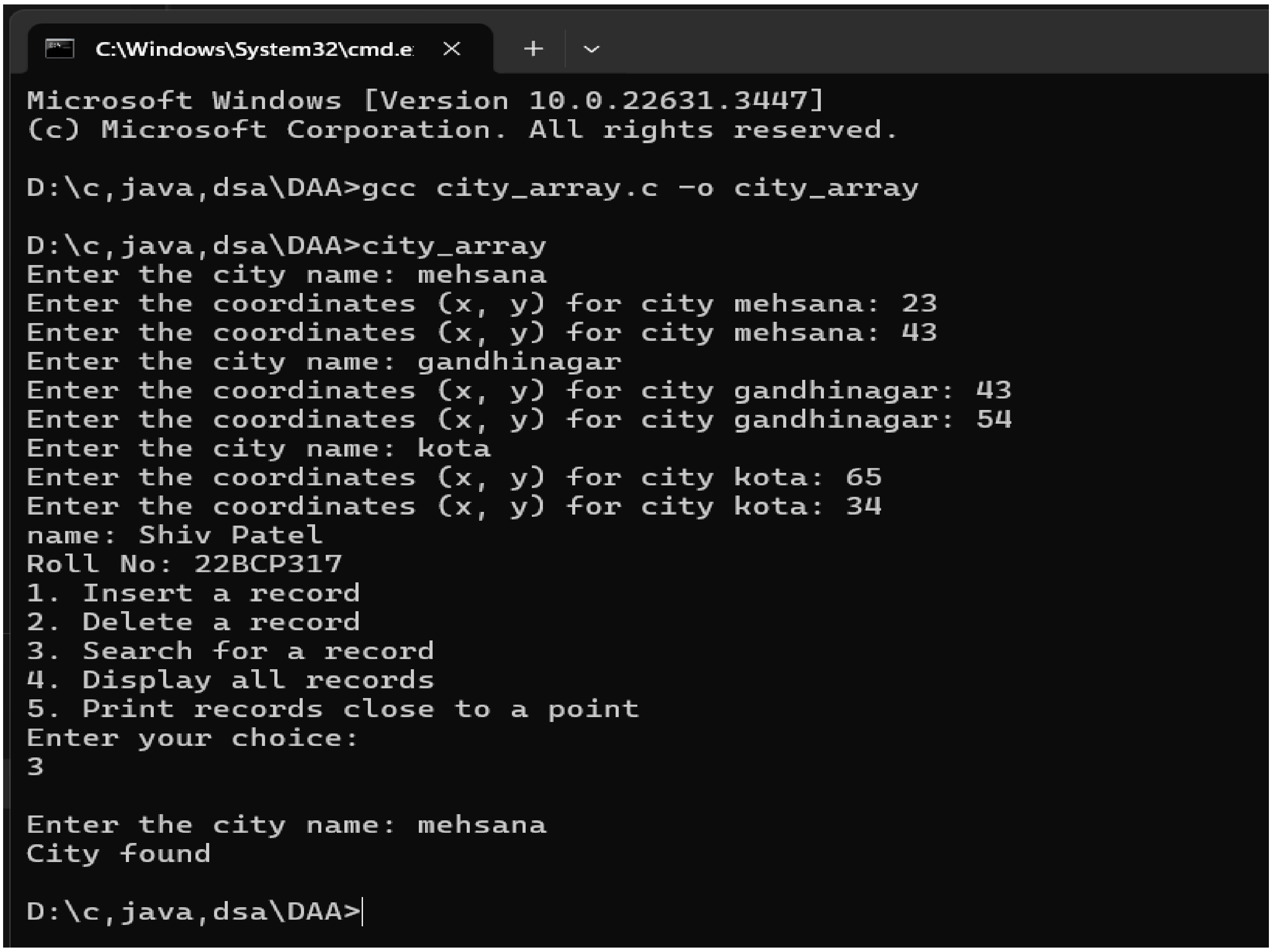
Source code:

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

int createRecords(char cityNames[][100], int numOfCities, double coordinates[][2]) {
   printf("Enter the records:\n");
   for (int i = 0; i < numOfCities; i++) {
      printf("Enter the city name: ");
      scanf("%s", cityNames[i]);
      for (int j = 0; j < 2; j++) {
         printf("Enter the coordinates (x, y) for city %s: ", cityNames[i]);
         scanf("%lf", &coordinates[i][j]);
      }
   }
   return numOfCities;
}

void displayRecords(char cityNames[][100], double coordinates[][2], int numOfCities) {
   printf("\nDisplaying records:\n");
   for (int i = 0; i < numOfCities; i++) {
      printf("City %d - Name: %s, Coordinates: (%lf, %lf)\n", i + 1, cityNames[i], coordinates[i][0],
coordinates[i][1]);
   }
}

int insertRecord(char cityNames[][100], double coordinates[][2], int numOfCities) {
   printf("\nEnter the city name: ");
   scanf("%s", cityNames[numOfCities]);
   for (int j = 0; j < 2; j++) {
      printf("Enter the coordinates (x, y) for city %s: ", cityNames[numOfCities]);
      scanf("%lf", &coordinates[numOfCities][j]);
   }
   return numOfCities + 1;
}

int deleteRecord(char cityNames[][100], double coordinates[][2], int numOfCities) {
   printf("\nEnter the city name: ");
   char cityName[100];
   scanf("%s", cityName);
   for (int i = 0; i < numOfCities; i++) {
      int value = strcmp(cityName, cityNames[i]);
      if (value == 0) {
         while (i < numOfCities) {
            strcpy(cityNames[i], cityNames[i + 1]);
```

```
                coordinates[i][0] = coordinates[i + 1][0];
                coordinates[i][1] = coordinates[i + 1][1];
                i++;
            }
        }
    }
    return numOfCities - 1;
}

void searchRecord(char cityNames[][100], double coordinates[][2], int numOfCities) {
    int found = 0;
    printf("\nEnter the city name: ");
    char cityName[100];
    scanf("%s", cityName);
    for (int i = 0; i < numOfCities; i++) {
        int value = strcmp(cityName, cityNames[i]);
        if (value == 0) {
            printf("City found\n");
            found = 1;
            break;
        }
    }
    if (found == 0) {
        printf("City not found\n");
    }
}

void printCloseRecords(char cityNames[][100], double coordinates[][2], int numOfCities) {
    double xCoord, yCoord, specifiedDistance;
    printf("Enter the specified distance point: ");
    scanf("%lf", &specifiedDistance);
    printf("Enter x coordinate of location: ");
    scanf("%lf", &xCoord);
    printf("Enter y coordinate of location: ");
    scanf("%lf", &yCoord);

    for (int i = 0; i < numOfCities; i++) {
        double xDiff = (xCoord - coordinates[i][0]) * (xCoord - coordinates[i][0]);
        double yDiff = (yCoord - coordinates[i][1]) * (yCoord - coordinates[i][1]);
        double distance = sqrt(xDiff + yDiff);
        if (distance <= specifiedDistance) {
            printf("%s\n", cityNames[i]);
        }
    }
}
```

```
int main() {
    char cityNames[100][100];
    double coordinates[100][2];
    int numOfCities = 3;
    int choice;

    numOfCities = createRecords(cityNames, numOfCities, coordinates);

    printf("Enter your choice:\n");
    printf("1. Insert a record\n");
    printf("2. Delete a record\n");
    printf("3. Search for a record\n");
    printf("4. Display all records\n");
    printf("5. Print records close to a point\n");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            numOfCities = insertRecord(cityNames, coordinates, numOfCities);
            break;
        case 2:
            numOfCities = deleteRecord(cityNames, coordinates, numOfCities);
            break;
        case 3:
            searchRecord(cityNames, coordinates, numOfCities);
            break;
        case 4:
            displayRecords(cityNames, coordinates, numOfCities);
            break;
        case 5:
            printCloseRecords(cityNames, coordinates, numOfCities);
            break;
        default:
            printf("Invalid choice\n");
            break;
    }

    Return 0;
}
```

Output:



C:\Windows\System32\cmd.e

```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

D:\c,java,dsa\DAA>gcc city_array.c -o city_array

D:\c,java,dsa\DAA>city_array
Enter the city name: mehsana
Enter the coordinates (x, y) for city mehsana: 23
Enter the coordinates (x, y) for city mehsana: 43
Enter the city name: gandhinagar
Enter the coordinates (x, y) for city gandhinagar: 43
Enter the coordinates (x, y) for city gandhinagar: 54
Enter the city name: kota
Enter the coordinates (x, y) for city kota: 65
Enter the coordinates (x, y) for city kota: 34
name: Shiv Patel
Roll No: 22BCP317
1. Insert a record
2. Delete a record
3. Search for a record
4. Display all records
5. Print records close to a point
Enter your choice:
3

Enter the city name: mehsana
City found

D:\c,java,dsa\DAA>
```

Linked List Implementation:

Memory Usage:
Linked lists tend to use more memory because each node in the list contains a pointer to the next node, along with the actual data. For a city database, this means additional memory overhead for storing pointers.
Insertion/Deletion:
Insertion and deletion operations can be more efficient in linked lists, especially if you're deleting or inserting elements in the middle of the list. This is because you only need to adjust pointers, rather than shifting elements in an array.
Search:
Searching for a record by name or coordinate might be less efficient in a linked list compared to an array. In an unsorted linked list, you would need to traverse the entire list to find the desired record, resulting in O(n) time complexity. However, if the linked list is sorted by name or coordinate, searching can be faster.
Printing Records Within a Given Distance: This operation can be relatively slower with linked lists because you need to traverse the entire list and calculate the distance for each city.

Array Implementation:

Memory Usage:
Arrays generally use less memory compared to linked lists because they store elements in contiguous memory locations. However, if you use a fixed-size array, you might encounter wasted space if the array is larger than needed.
Insertion/Deletion:
Insertion and deletion operations can be less efficient in arrays, especially for large datasets, because you may need to shift elements to accommodate the changes.
Search:
Searching for a record by name or coordinate can be more efficient in an array if the array is sorted. With a sorted array, you can use binary search, which has a time complexity of O(log n). However, if the array is unsorted, searching would still require traversing the entire array, resulting in O(n) time complexity.
Printing Records Within a Given Distance: This operation can be relatively faster with arrays if the array is sorted by coordinates. You can use binary search to find the range of cities within the specified distance.

5	**[Greedy Approach]** Implement interval scheduling algorithm. Given n events with their starting and ending times, find a schedule that includes as many events as possible. It is not possible to select an event partially. For example, consider the following example:

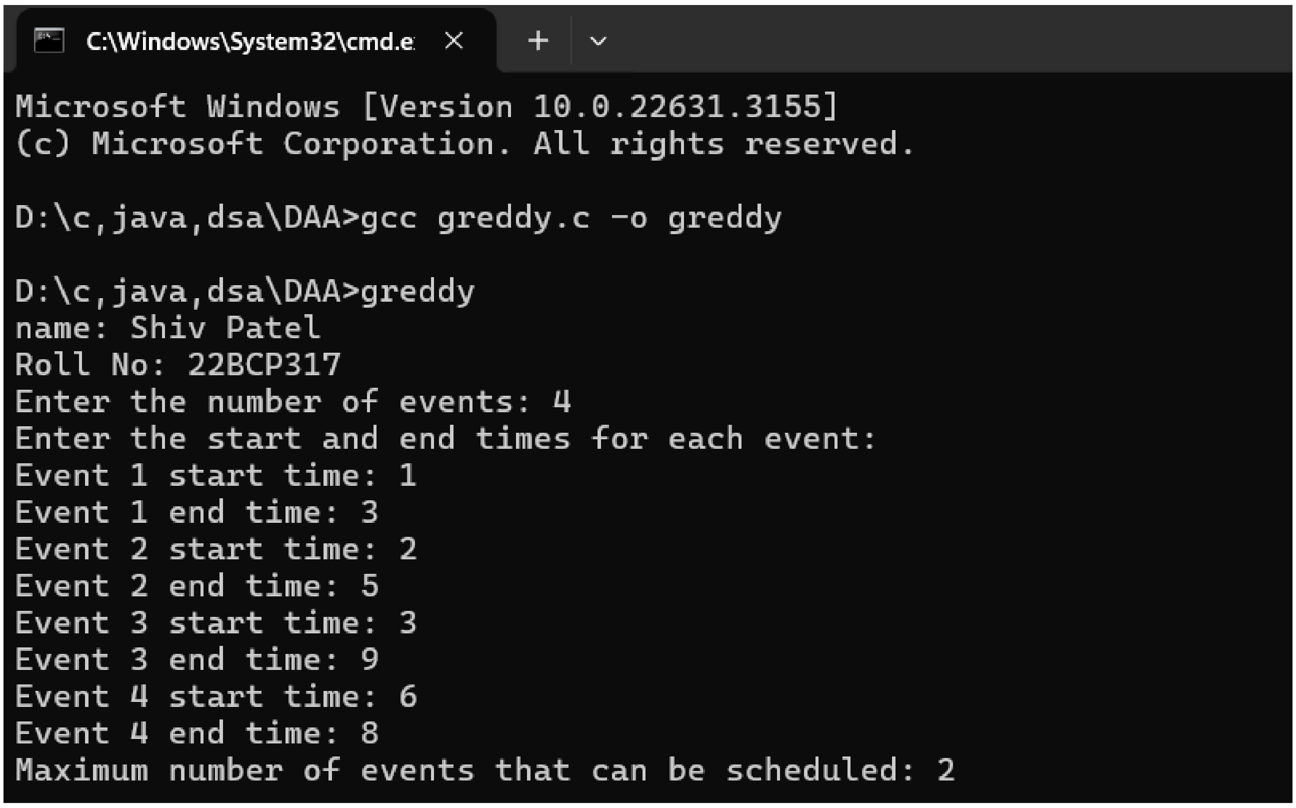
Event	Starting time	Ending time
A	1	3
B	2	5
C	3	9
D	6	8

Here, maximum number of events that can be scheduled is 2. We can schedule B and D together.

Source code:

```
#include <stdio.h>

int findMaxEvents(int startTimes[], int endTimes[], int n) {
    int maxEvents = 0;
    int total = 0;

    for (int i = 0; i < n; i++) {
        if (startTimes[i] >= total) {
            total = endTimes[i];
            maxEvents++;
        }
    }
    return maxEvents;
}

int main() {
    int n;
    printf("name: Shiv Patel\n");
    printf("Roll No: 22BCP317\n");

    printf("Enter the number of events: ");
    scanf("%d", &n);

    int startTimes[50], endTimes[50];
    printf("Enter the start and end times for each event:\n");
    for (int i = 0; i < n; i++) {
        printf("Event %d start time: ", i + 1);
        scanf("%d", &startTimes[i]);
        printf("Event %d end time: ", i + 1);
        scanf("%d", &endTimes[i]);
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (endTimes[j] > endTimes[j + 1]) {
                int temp = endTimes[j];
                endTimes[j] = endTimes[j + 1];
                endTimes[j + 1] = temp;

                int temp2 = startTimes[j];
                startTimes[j] = startTimes[j + 1];
                startTimes[j + 1] = temp2;
            }
        }
    }

    int maxEvents = findMaxEvents(startTimes, endTimes, n);
    printf("Maximum number of events that can be scheduled: %d\n", maxEvents);

    return 0;
}
```

OUTPUT:



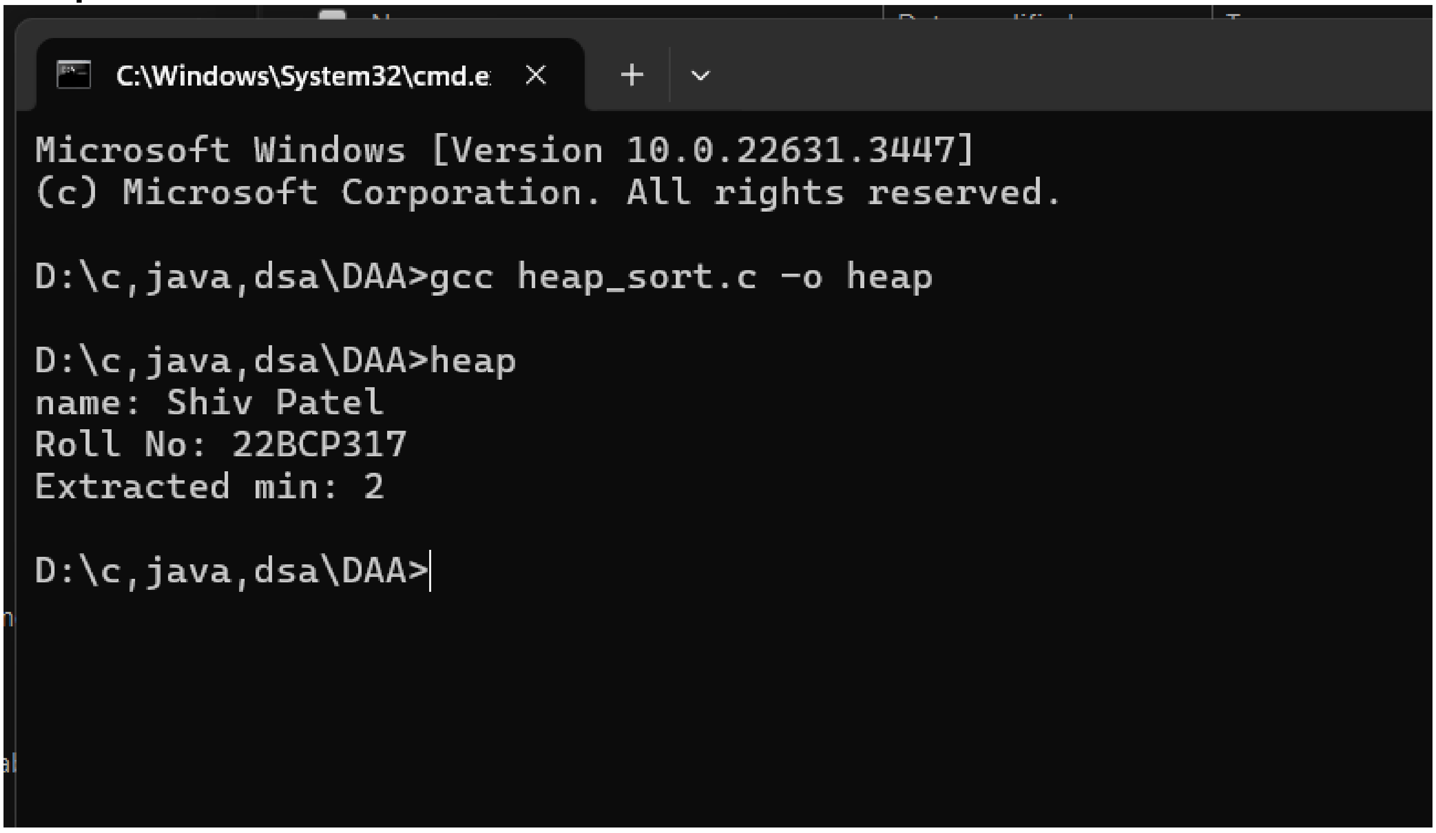
```
Microsoft Windows [Version 10.0.22631.3155]
(c) Microsoft Corporation. All rights reserved.

D:\c,java,dsa\DAA>gcc greddy.c -o greddy

D:\c,java,dsa\DAA>greddy
name: Shiv Patel
Roll No: 22BCP317
Enter the number of events: 4
Enter the start and end times for each event:
Event 1 start time: 1
Event 1 end time: 3
Event 2 start time: 2
Event 2 end time: 5
Event 3 start time: 3
Event 3 end time: 9
Event 4 start time: 6
Event 4 end time: 8
Maximum number of events that can be scheduled: 2
```

Min heap:

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

struct min_heap
{
   int *array;
   int capacity;
   int head;
};

struct min_heap *create_min_heap(int capacity)
{
   struct min_heap *min_heap = (struct min_heap *)malloc(sizeof(struct min_heap));
   min_heap->capacity = capacity;
   min_heap->head = 0;
   min_heap->array = (int *)malloc(min_heap->capacity * sizeof(int));
   return min_heap;
}

void swap(int *a, int *b)
{
   int temp = *a;
   *a = *b;
   *b = temp;
}

void min_heapify(struct min_heap *min_heap, int index)
{
   int smallest = index;
   int left = 2 * index + 1 ;
   int right = 2 * index + 2 ;

   if (left < min_heap->head && min_heap->array[left] < min_heap->array[smallest])
      smallest = left;

   if (right < min_heap->head && min_heap->array[right] < min_heap->array[smallest])
      smallest = right;

   if (smallest != index)
   {
      swap(&min_heap->array[index], &min_heap->array[smallest]);
      min_heapify(min_heap, smallest);
   }
}

void insert_key(struct min_heap *min_heap, int key)
{
   if (min_heap->head == min_heap->capacity)
   {
      printf("Overflow: Could not insert key\n");
      return;
   }

   int i = min_heap->head;
```

```
    min_heap->array[i] = key;
    min_heap->head++;

    while (i != 0 && min_heap->array[(i - 1) / 2] > min_heap->array[i])
    {
        swap(&min_heap->array[i], &min_heap->array[(i - 1) / 2]);
        i = (i - 1) / 2;
    }
}

int extract_min(struct min_heap *min_heap)
{
    if (min_heap->head <= 0)
        return 0;

    if (min_heap->head == 1)
    {
        min_heap->head--;
        return min_heap->array[0];
    }

    int root = min_heap->array[0];
    min_heap->array[0] = min_heap->array[min_heap->head - 1];
    min_heap->head--;
    min_heapify(min_heap, 0);

    return root;
}

int main()
{
    struct min_heap *min_heap = create_min_heap(100);

    insert_key(min_heap, 6);
    insert_key(min_heap, 2);
    insert_key(min_heap, 7);
    insert_key(min_heap, 8);
    insert_key(min_heap, 4);

    printf("name: Shiv Patel\n");
    printf("Roll No: 22BCP317\n");
    printf("Extracted min: %d\n", extract_min(min_heap));

    free(min_heap->array);
    free(min_heap);

    return 0;
}
```

Output:



```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

D:\c,java,dsa\DAA>gcc heap_sort.c -o heap

D:\c,java,dsa\DAA>heap
name: Shiv Patel
Roll No: 22BCP317
Extracted min: 2

D:\c,java,dsa\DAA>
```

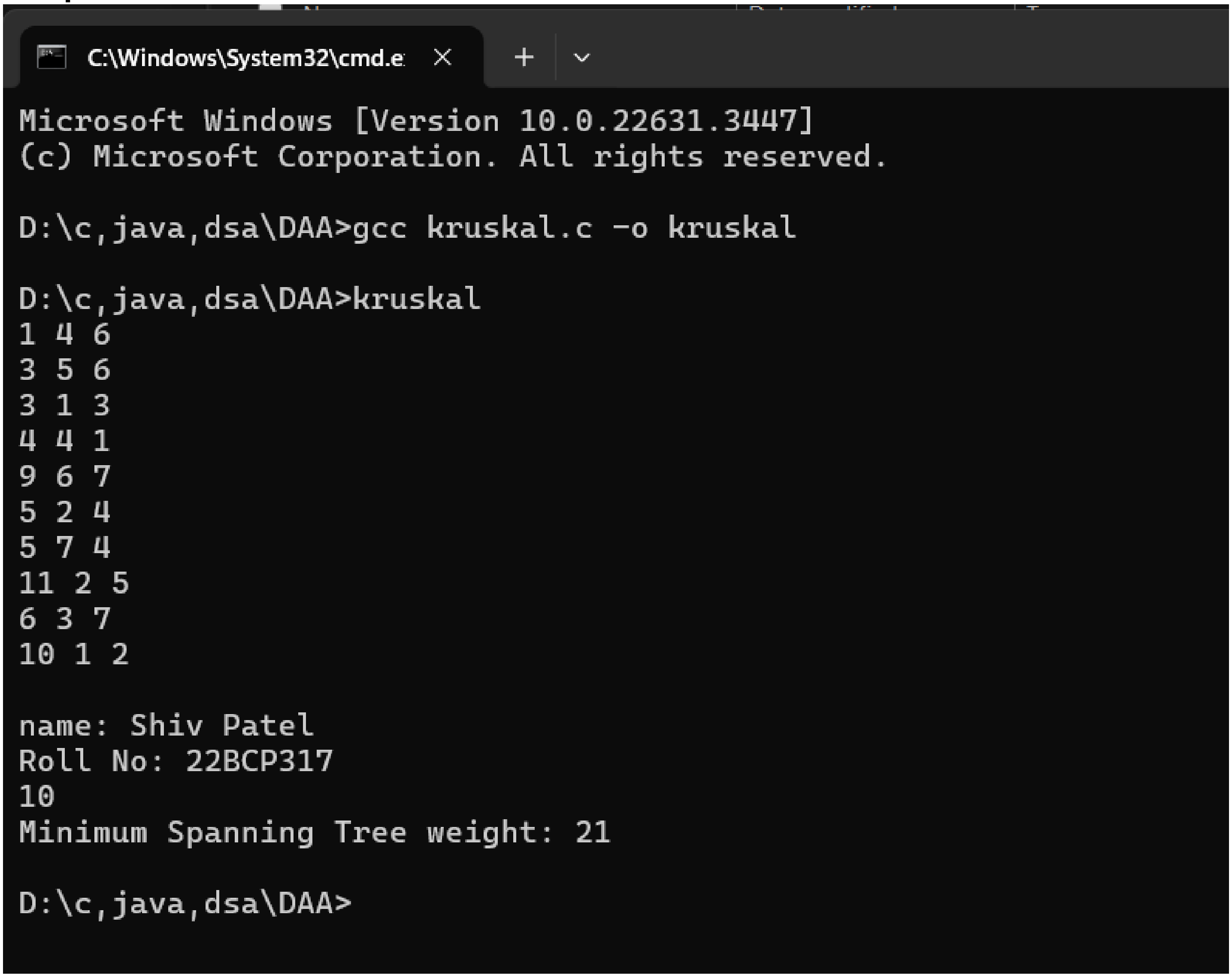
Kruskal algorithm:

Source code:

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


struct Node
{
   int weight;
   int src; // source
   int des; // destination
};

struct MinHeap
{
   int size;
   int capacity;
   struct Node *array;
};

struct Node *createNode(int weight, int src, int des)
{
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->weight = weight;
   newNode->src = src;
   newNode->des = des;
   return newNode;
}

struct MinHeap *createMinHeap(int capacity)
{
   struct MinHeap *minHeap = (struct MinHeap *)malloc(sizeof(struct MinHeap));
   minHeap->size = 0;
   minHeap->capacity = capacity;
   minHeap->array = (struct Node *)malloc(capacity * sizeof(struct Node));
   return minHeap;
}

void swapNodes(struct Node *a, struct Node *b)
{
   struct Node temp = *a;
   *a = *b;
   *b = temp;
}

void minHeapify(struct MinHeap *minHeap, int idx)
{
   int smallest = idx;
   int left = 2 * idx + 1;
   int right = 2 * idx + 2;
```

```
    if (left < minHeap->size && minHeap->array[left].weight < minHeap->array[smallest].weight)
        smallest = left;

    if (right < minHeap->size && minHeap->array[right].weight < minHeap->array[smallest].weight)
        smallest = right;

    if (smallest != idx)
    {
        swapNodes(&minHeap->array[smallest], &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}

void insert(struct MinHeap *minHeap, int weight, int src, int des)
{
    if (minHeap->size == minHeap->capacity)
    {
        printf("Heap overflow\n");
        return;
    }

    int i = minHeap->size;
    minHeap->array[i].weight = weight;
    minHeap->array[i].src = src;
    minHeap->array[i].des = des;
    minHeap->size++;

    while (i && minHeap->array[i].weight < minHeap->array[(i - 1) / 2].weight)
    {
        swapNodes(&minHeap->array[i], &minHeap->array[(i - 1) / 2]);
        i = (i - 1) / 2;
    }
}

struct Node extractMin(struct MinHeap *minHeap)
{
    if (minHeap->size <= 0)
    {
        struct Node nullNode = {-1, -1, -1};
        return nullNode;
    }
    if (minHeap->size == 1)
    {
        minHeap->size--;
        return minHeap->array[0];
    }

    struct Node root = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    minHeap->size--;
    minHeapify(minHeap, 0);
    return root;
}

void printHeap(struct MinHeap *minHeap)
{
    for (int i = 0; i < minHeap->size; ++i)
    {
        printf("%d %d %d\n", minHeap->array[i].weight, minHeap->array[i].src, minHeap-
>array[i].des);
    }
    printf("\n");
}

int findRootParent(int node, int parent[])
{
    if (node == parent[node])
    {
        return node;
    }
    return parent[node] = findRootParent(parent[node], parent);
}

void unionBySize(int u, int v, int parent[], int size[])
{
    int rootParentU = findRootParent(u, parent);
    int rootParentV = findRootParent(v, parent);
    if (rootParentU != rootParentV)
    {
        if (size[rootParentU] < size[rootParentV])
        {
            parent[rootParentU] = rootParentV;
            size[rootParentV] += size[rootParentU];
        }
        else
        {
            parent[rootParentV] = rootParentU;
            size[rootParentU] += size[rootParentV];
        }
    }
}

int main()
{
    int size = 0;
    int parent[HEAP_SIZE];
    int sizeArray[HEAP_SIZE];
    for (int i = 0; i <= HEAP_SIZE; i++)
    {
        parent[i] = i;
        sizeArray[i] = 1;
    }

    struct MinHeap *minHeap = createMinHeap(HEAP_SIZE);
    insert(minHeap, 3, 1, 3);
    insert(minHeap, 6, 3, 7);
    insert(minHeap, 5, 7, 4);
    insert(minHeap, 4, 4, 1);
    insert(minHeap, 10, 1, 2);
    insert(minHeap, 5, 2, 4);
    insert(minHeap, 1, 4, 6);
    insert(minHeap, 11, 2, 5);
    insert(minHeap, 3, 5, 6);
    insert(minHeap, 9, 6, 7);

    printHeap(minHeap);
    printf("name: Shiv Patel\n");
    printf("Roll No: 22BCP317\n");
    int N = 7;
    printf("%d\n", minHeap->size);
    int mst = 0;
    while (N > 1)
    {
        struct Node minNode = extractMin(minHeap);
        int weight = minNode.weight;
        int src = minNode.src;
        int des = minNode.des;
        unionBySize(src, des, parent, sizeArray);
        mst += weight;
        N--;
    }

    printf("Minimum Spanning Tree weight: %d\n", mst);

    free(minHeap->array);
    free(minHeap);

    return 0;
}
```

Output:



C:\Windows\System32\cmd.e

```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

D:\c,java,dsa\DAA>gcc kruskal.c -o kruskal

D:\c,java,dsa\DAA>kruskal
1 4 6
3 5 6
3 1 3
4 4 1
9 6 7
5 2 4
5 7 4
11 2 5
6 3 7
10 1 2

name: Shiv Patel
Roll No: 22BCP317
10
Minimum Spanning Tree weight: 21

D:\c,java,dsa\DAA>
```

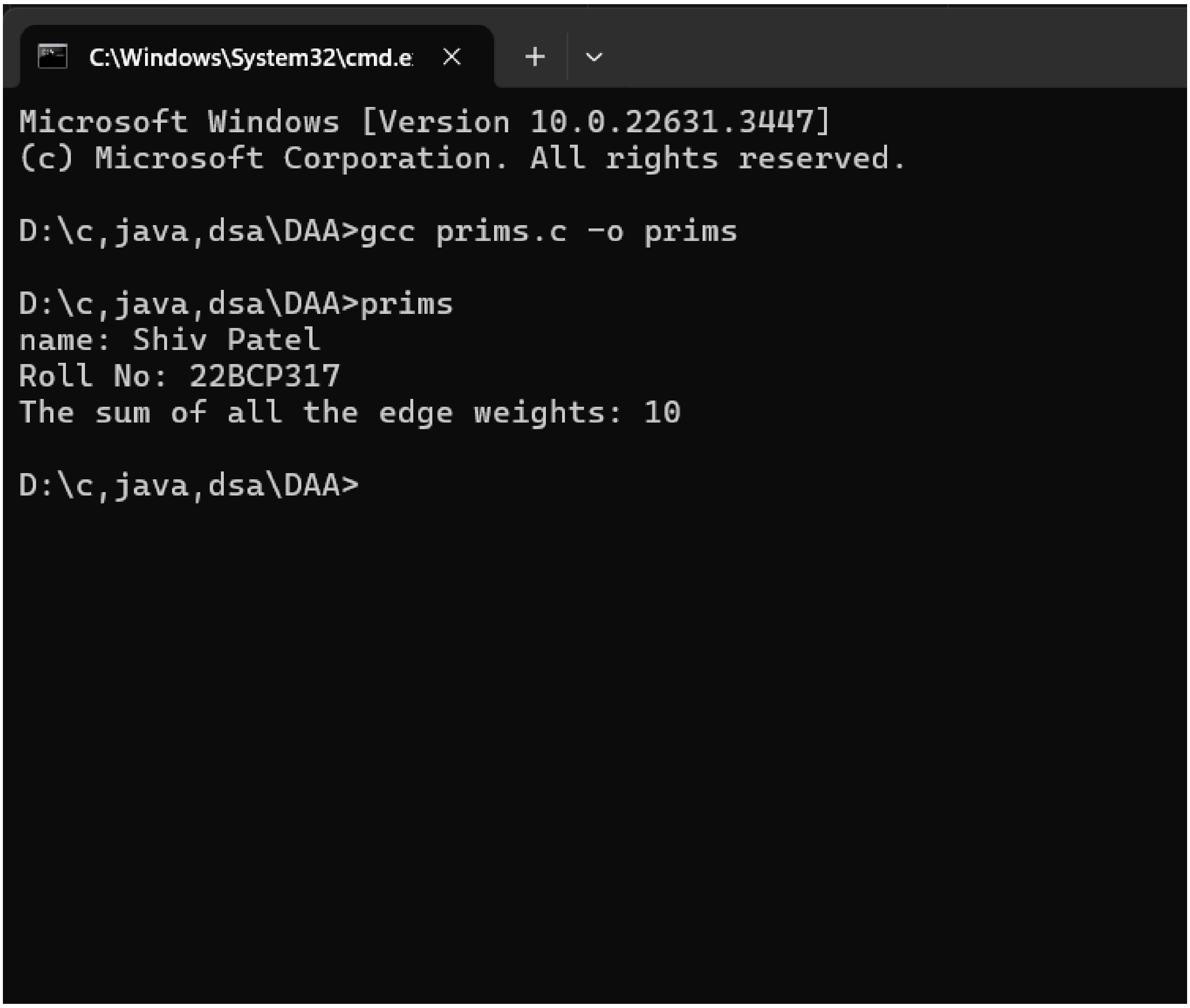
Prims algorithm:

Source code:

```
#include <stdio.h>

void heapify(int heap[100][3], int size, int index)
{
   int smallest = index;
   int left = 2 * index + 1;
   int right = 2 * index + 2;

   if (left < size && heap[left][0] < heap[smallest][0])
      smallest = left;

   if (right < size && heap[right][0] < heap[smallest][0])
      smallest = right;

   if (smallest != index)
   {
      int tempWeight = heap[index][0];
      int tempSrc = heap[index][1];
      int tempDes = heap[index][2];
      heap[index][0] = heap[smallest][0];
      heap[index][1] = heap[smallest][1];
      heap[index][2] = heap[smallest][2];
      heap[smallest][0] = tempWeight;
      heap[smallest][1] = tempSrc;
      heap[smallest][2] = tempDes;
      heapify(heap, size, smallest);
   }
}

void buildHeap(int heap[100][3], int size)
{
   for (int i = size / 2 - 1; i >= 0; i--)
   {
      heapify(heap, size, i);
   }
}

void insert(int heap[100][3], int *size, int weight, int src, int des)
{
   (*size)++;
   int i = *size - 1;
   heap[i][0] = weight;
   heap[i][1] = src;
   heap[i][2] = des;
   while (i != 0 && heap[(i - 1) / 2][0] > heap[i][0])
   {
      int tempWeight = heap[i][0];
      int tempSrc = heap[i][1];
      int tempDes = heap[i][2];
      heap[i][0] = heap[(i - 1) / 2][0];
      heap[i][1] = heap[(i - 1) / 2][1];
      heap[i][2] = heap[(i - 1) / 2][2];
      heap[(i - 1) / 2][0] = tempWeight;
      heap[(i - 1) / 2][1] = tempSrc;
```

```
        heap[(i - 1) / 2][2] = tempDes;
        i = (i - 1) / 2;
    }
}

void removeMin(int heap[100][3], int *size)
{
    if (*size <= 0)
    {
        printf("Nothing to delete\n");
        return;
    }
    if (*size == 1)
    {
        (*size)--;
        return;
    }

    heap[0][0] = heap[*size - 1][0];
    heap[0][1] = heap[*size - 1][1];
    heap[0][2] = heap[*size - 1][2];
    (*size)--;
    heapify(heap, *size, 0);
}

int spanningTree(int V, int adj[][100][2])
{
    int heap[100][3];
    int size = 0;
    int vis[100] = {0};
    int sum = 0;

    insert(heap, &size, 0, 0, 0);
    while (size > 0)
    {
        int node = heap[0][2];
        int wt = heap[0][0];
        removeMin(heap, &size);

        if (vis[node])
            continue;

        vis[node] = 1;
        sum += wt;

        for (int i = 0; i < V; ++i)
        {
            if (adj[node][i][0] != -1)
            {
                int adjNode = adj[node][i][0];
                int edW = adj[node][i][1];
                if (!vis[adjNode])
                {
                    insert(heap, &size, edW, node, adjNode);
                }
            }
        }
    }
    return sum;
}

int main()
{
    int V = 5;
    int edges[][3] = {{0, 1, 2}, {0, 2, 1}, {1, 2, 1}, {2, 3, 2}, {3, 4, 1}, {4, 2, 2}};
    int adj[5][100][2];
    for (int i = 0; i < V; ++i)
    {
        for (int j = 0; j < 100; ++j)
        {
            adj[i][j][0] = -1;
            adj[i][j][1] = -1;
        }
    }
    for (int i = 0; i < 6; ++i)
    {
        int node1 = edges[i][0];
        int node2 = edges[i][1];
        int weight = edges[i][2];
        for (int j = 0; j < 100; ++j)
        {
            if (adj[node1][j][0] == -1)
            {
                adj[node1][j][0] = node2;
                adj[node1][j][1] = weight;
                break;
            }
        }
        for (int j = 0; j < 100; ++j)
        {
            if (adj[node2][j][0] == -1)
            {
                adj[node2][j][0] = node1;
                adj[node2][j][1] = weight;
                break;
            }
        }
    }

    printf("name: Shiv Patel\n");
    printf("Roll No: 22BCP317\n");
    int sum = spanningTree(V, adj);
    printf("The sum of all the edge weights: %d\n", sum);

    return 0;
}
```

Output:



```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

D:\c,java,dsa\DAA>gcc prims.c -o prims

D:\c,java,dsa\DAA>prims
name: Shiv Patel
Roll No: 22BCP317
The sum of all the edge weights: 10

D:\c,java,dsa\DAA>
```

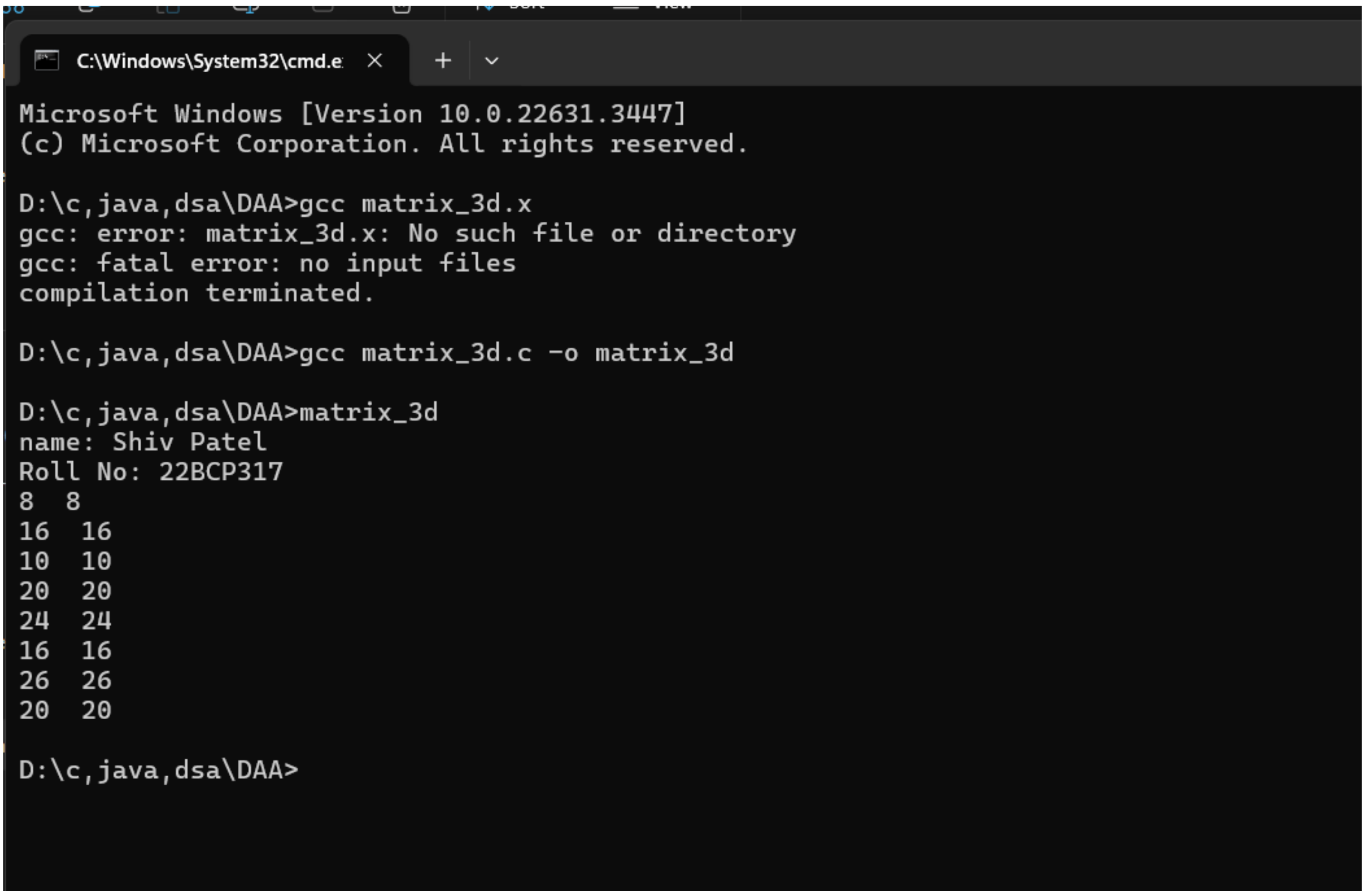
6	**[Divide and Conquer]** Implement both a standard $O(n^3)$ matrix multiplication algorithm and Strassen's matrix multiplication algorithm. Using empirical testing, try and estimate the constant factors for the runtime equations of the two algorithms. How big must n be before Strassen's algorithm becomes more efficient than the standard algorithm?		

Source code:

```
#include <stdio.h>

void multiply(int A[][2], int B[][2], int C[][2])
{
   for (int i = 0; i < 2; i++)
   {
      for (int j = 0; j < 2; j++)
      {
         C[i][j] = 0;
         for (int k = 0; k < 2; k++)
         {
            C[i][j] += A[i][k] * B[k][j];
         }
      }
   }
}

void add(int A[][2], int B[][2], int C[][2])
{
   for (int i = 0; i < 2; i++)
   {
      for (int j = 0; j < 2; j++)
      {
         C[i][j] = A[i][j] + B[i][j];
      }
   }
}
void print(int A[][2])
{
   for (int i = 0; i < 2; i++)
   {
      for (int j = 0; j < 2; j++)
      {
         printf("%d  ", A[i][j]);
      }
      printf("\n");
   }
}
void sub(int A[][2], int B[][2], int C[][2])
{
   for (int i = 0; i < 2; i++)
   {
      for (int j = 0; j < 2; j++)
      {
        C[i][j] = A[i][j] - B[i][j];
      }
   }
}

int main()
{
   int a11[2][2], b11[2][2], a22[2][2], b22[2][2], c1[2][2], c2[2][2], m1[2][2];
   int a12[2][2], b12[2][2], a21[2][2], b21[2][2];
   int m2[2][2], m3[2][2], m4[2][2], m5[2][2], m6[2][2], m7[2][2];
   int d11[2][2], d12[2][2], d21[2][2], d22[2][2];

   int a[4][4] = {{1, 1, 1, 1}, {2, 2, 2, 2}, {3, 3, 3, 3}, {2, 2, 2, 2}};

   int b[4][4] = {{1, 1, 1, 1}, {2, 2, 2, 2}, {3, 3, 3, 3}, {2, 2, 2, 2}};

   for (int i = 0; i < 2; i++)
   {
      for (int j = 0; j < 2; j++)
      {
         a11[i][j] = a[i][j];
         b11[i][j] = b[i][j];
         a12[i][j] = a[i][j + 2];
         b12[i][j] = b[i][j + 2];
         a21[i][j] = a[i + 2][j];
         b21[i][j] = b[i + 2][j];
         a22[i][j] = a[i + 2][j + 2];
         b22[i][j] = b[i + 2][j + 2];
      }
   }
   add(a11, a22, c1);
   add(b11, b22, c2);
   multiply(c1, c2, m1); // m1

   add(a21, a22, c1);
   multiply(c1, b11, m2); // m2

   sub(b21, b22, c2);
   multiply(a11, c2, m3); // m3

   sub(b21, b11, c1);
   multiply(a22, c1, m4); // m4

   add(a11, a12, c1);
   multiply(c1, b22, m5); // m5

   sub(a21, a11, c1);
   add(b11, b12, c2);
   multiply(c1, c2, m6); // m6

   sub(a12, a22, c1);
   add(b21, b22, c2);
   multiply(c1, c2, m7); // m7
```

```
    add(m3, m5, d12);
    add(m2, m4, d21);

    add(m1, m4, c1);
    add(c1, m7, c2);
    sub(c2, m5, d11);

    sub(m1, m2, c2);
    add(c2, m3, c1);
    add(c1, m6, d22);
    printf("name: Shiv Patel\n");
    printf("Roll No: 22BCP317\n");
    print(d11);
    print(d12);
    print(d21);
    print(d22);

    return 0;
}
```

Output:



```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

D:\c,java,dsa\DAA>gcc matrix_3d.x
gcc: error: matrix_3d.x: No such file or directory
gcc: fatal error: no input files
compilation terminated.

D:\c,java,dsa\DAA>gcc matrix_3d.c -o matrix_3d

D:\c,java,dsa\DAA>matrix_3d
name: Shiv Patel
Roll No: 22BCP317
8  8
16  16
10  10
20  20
24  24
16  16
26  26
20  20

D:\c,java,dsa\DAA>
```

7	**[Dynamic Programming]** mplement the Floyd Warshall Algorithm for All Pair Shortest Path Problem. You are given a weighted diagraph $G = (V, E)$, with arbitrary edge weights or costs c_{vw} between any node v and node w. Find the cheapest path from every node to every other node. Edges may have negative weights. Consider the following test case to check your algorithm:		

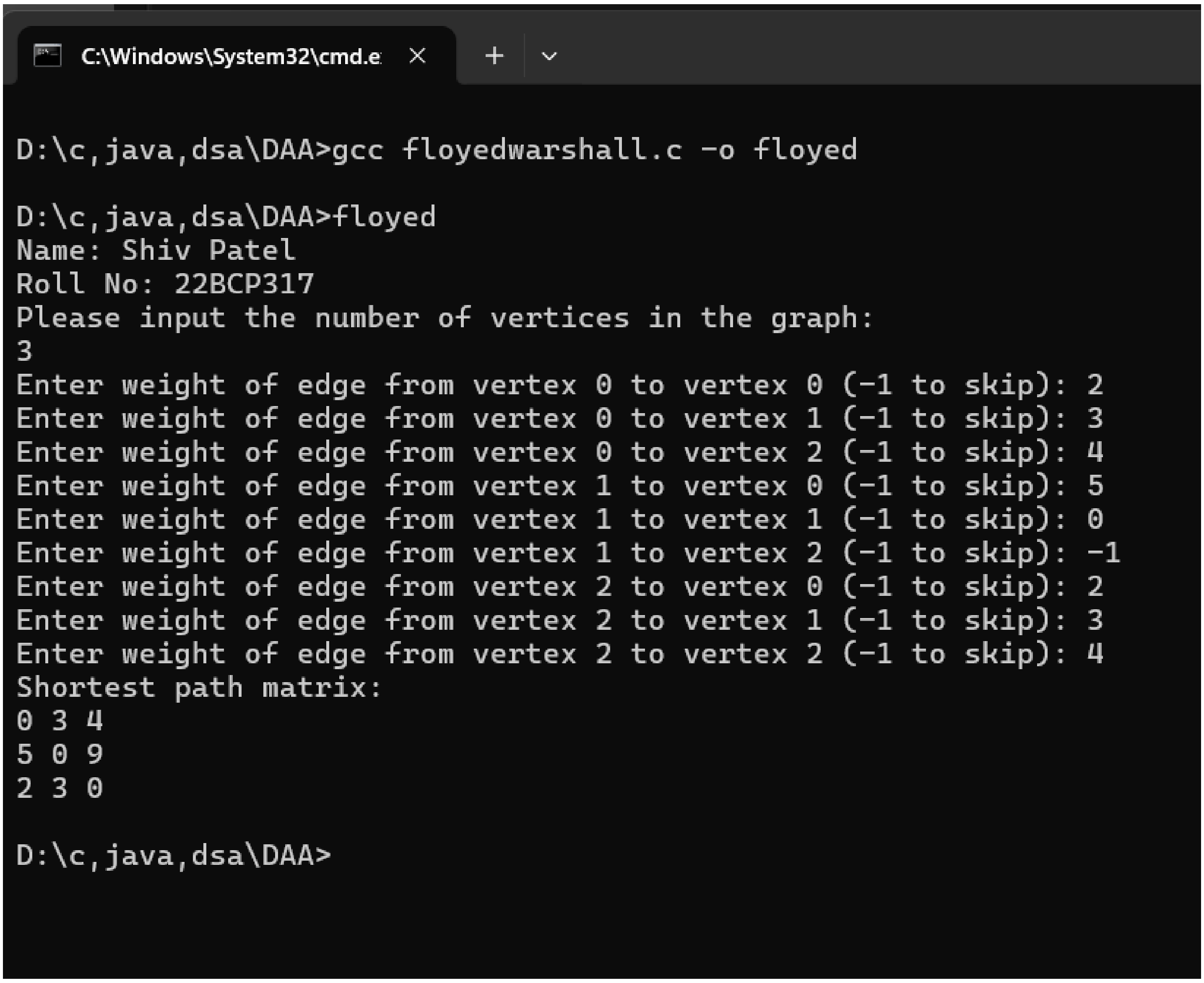
$\boldsymbol{v}$	$\boldsymbol{w}$	c_{vw}
0	1	-1
0	2	4
1	2	3
1	3	2
1	4	2
3	2	5
3	1	1
4	3	-3

Source code:

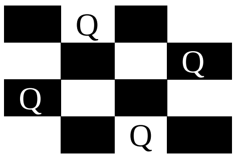
```
#include <stdio.h>

int main()
{
   int numVertices;
   printf("Name: Shiv Patel\n");
   printf("Roll No: 22BCP317\n");

   printf("Please input the number of vertices in the graph: \n");
   scanf("%d", &numVertices);

   int graph[numVertices][numVertices];
   for (int i = 0; i < numVertices; i++)
   {
      for (int j = 0; j < numVertices; j++)
      {
         int weight;
         printf("Enter weight of edge from vertex %d to vertex %d (-1 to skip): ", i, j);
         scanf("%d", &weight);
         graph[i][j] = weight;
      }
   }

   for (int i = 0; i < numVertices; i++)
   {
      for (int j = 0; j < numVertices; j++)
      {
         if (graph[i][j] == -1)
            graph[i][j] = INFINITY_VAL;
         if (i == j)
            graph[i][j] = 0;
```

```
        }
    }

    for (int k = 0; k < numVertices; k++)
    {
        for (int i = 0; i < numVertices; i++)
        {
            for (int j = 0; j < numVertices; j++)
            {
                if (graph[i][k] < INFINITY_VAL && graph[k][j] < INFINITY_VAL)
                {
                    if (graph[i][j] > graph[i][k] + graph[k][j])
                    {
                        graph[i][j] = graph[i][k] + graph[k][j];
                    }
                }
            }
        }
    }

    printf("Shortest path matrix:\n");
    for (int i = 0; i < numVertices; i++)
    {
        for (int j = 0; j < numVertices; j++)
        {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Output:



```
D:\c,java,dsa\DAA>gcc floyedwarshall.c -o floyed

D:\c,java,dsa\DAA>floyed
Name: Shiv Patel
Roll No: 22BCP317
Please input the number of vertices in the graph:
3
Enter weight of edge from vertex 0 to vertex 0 (-1 to skip): 2
Enter weight of edge from vertex 0 to vertex 1 (-1 to skip): 3
Enter weight of edge from vertex 0 to vertex 2 (-1 to skip): 4
Enter weight of edge from vertex 1 to vertex 0 (-1 to skip): 5
Enter weight of edge from vertex 1 to vertex 1 (-1 to skip): 0
Enter weight of edge from vertex 1 to vertex 2 (-1 to skip): -1
Enter weight of edge from vertex 2 to vertex 0 (-1 to skip): 2
Enter weight of edge from vertex 2 to vertex 1 (-1 to skip): 3
Enter weight of edge from vertex 2 to vertex 2 (-1 to skip): 4
Shortest path matrix:
0 3 4
5 0 9
2 3 0

D:\c,java,dsa\DAA>
```

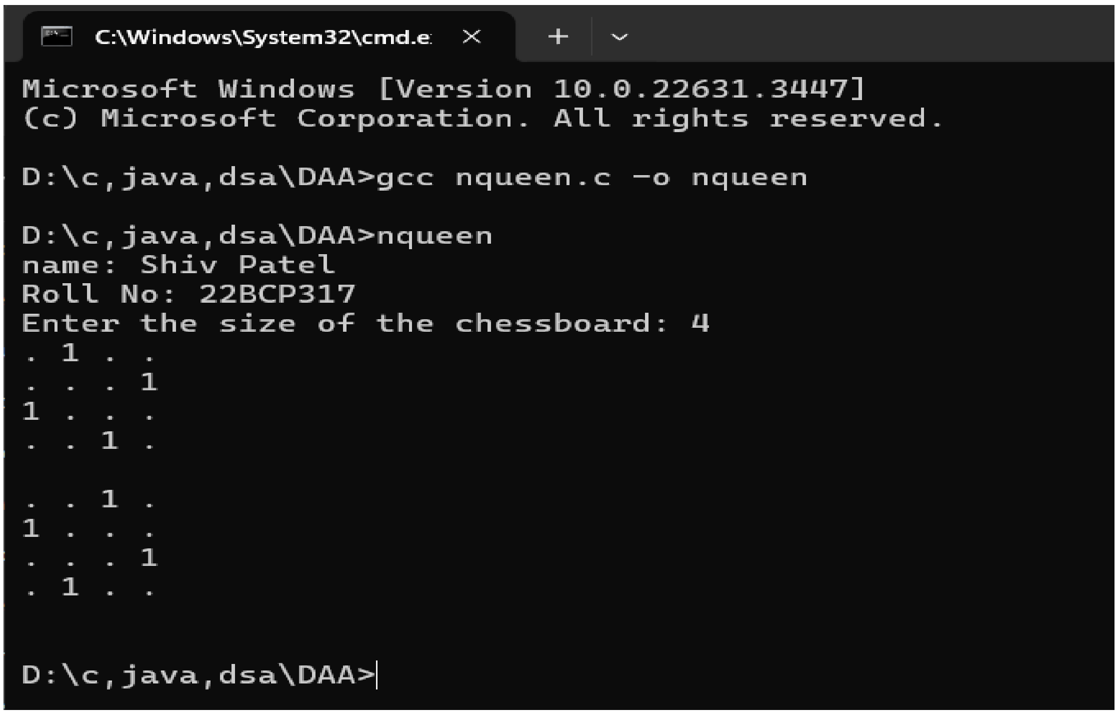
8	**[Backtracking]** Solve the n queens' problem using backtracking. Here, the task is to place n chess queens on an n x n board so that no two queens attack each other. For example, following is a solution for the 4 Queen' problem. 

Source code:

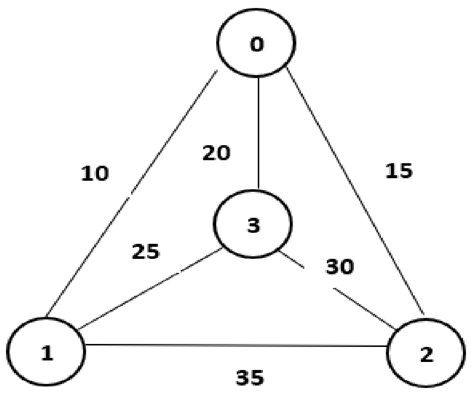
```
#include <stdio.h>
#include <stdlib.h>

#define MAXN 20

int isSafePosition(int i, int j, char chessBoard[MAXN][MAXN], int n)
{
   int x, y;

   for (x = 0; x < i; x++)
   {
      if (chessBoard[x][j] == '1')
         return 0;
   }

   for (x = i, y = j; x >= 0 && y >= 0; x--, y--)
   {
      if (chessBoard[x][y] == '1')
         return 0;
   }

   for (x = i, y = j; x >= 0 && y < n; x--, y++)
   {
      if (chessBoard[x][y] == '1')
         return 0;
   }

   return 1;
}

void printChessBoard(char chessBoard[MAXN][MAXN], int n)
{
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
      {
         printf("%c ", chessBoard[i][j]);
      }
      printf("\n");
```

```
    }
    printf("\n");
}

void solveHelper(int i, int n, char chessBoard[MAXN][MAXN])
{
    int j;
    if (i == n)
    {
        printChessBoard(chessBoard, n);
        return;
    }
    for (j = 0; j < n; j++)
    {
        if (isSafePosition(i, j, chessBoard, n))
        {
            chessBoard[i][j] = '1';
            solveHelper(i + 1, n, chessBoard);
            chessBoard[i][j] = '.';
        }
    }
}

void solveNQueens(int n)
{
    char chessBoard[MAXN][MAXN];

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            chessBoard[i][j] = '.';
        }
    }
    solveHelper(0, n, chessBoard);
}

int main()
{
    int n;
    printf("name: Shiv Patel\n");
    printf("Roll No: 22BCP317\n");

    printf("Enter the size of the chessboard: ");
    scanf("%d", &n);

    if (n < 1 || n > MAXN)
    {
        printf("Invalid size of the chessboard.\n");
        return 1;
    }
    solveNQueens(n);

    return 0;
}
```

Output:



```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

D:\c,java,dsa\DAA>gcc nqueen.c -o nqueen

D:\c,java,dsa\DAA>nqueen
name: Shiv Patel
Roll No: 22BCP317
Enter the size of the chessboard: 4
. 1 . .
. . . 1
1 . . .
. . 1 .

. . 1 .
1 . . .
. . . 1
. 1 . .


D:\c,java,dsa\DAA>
```

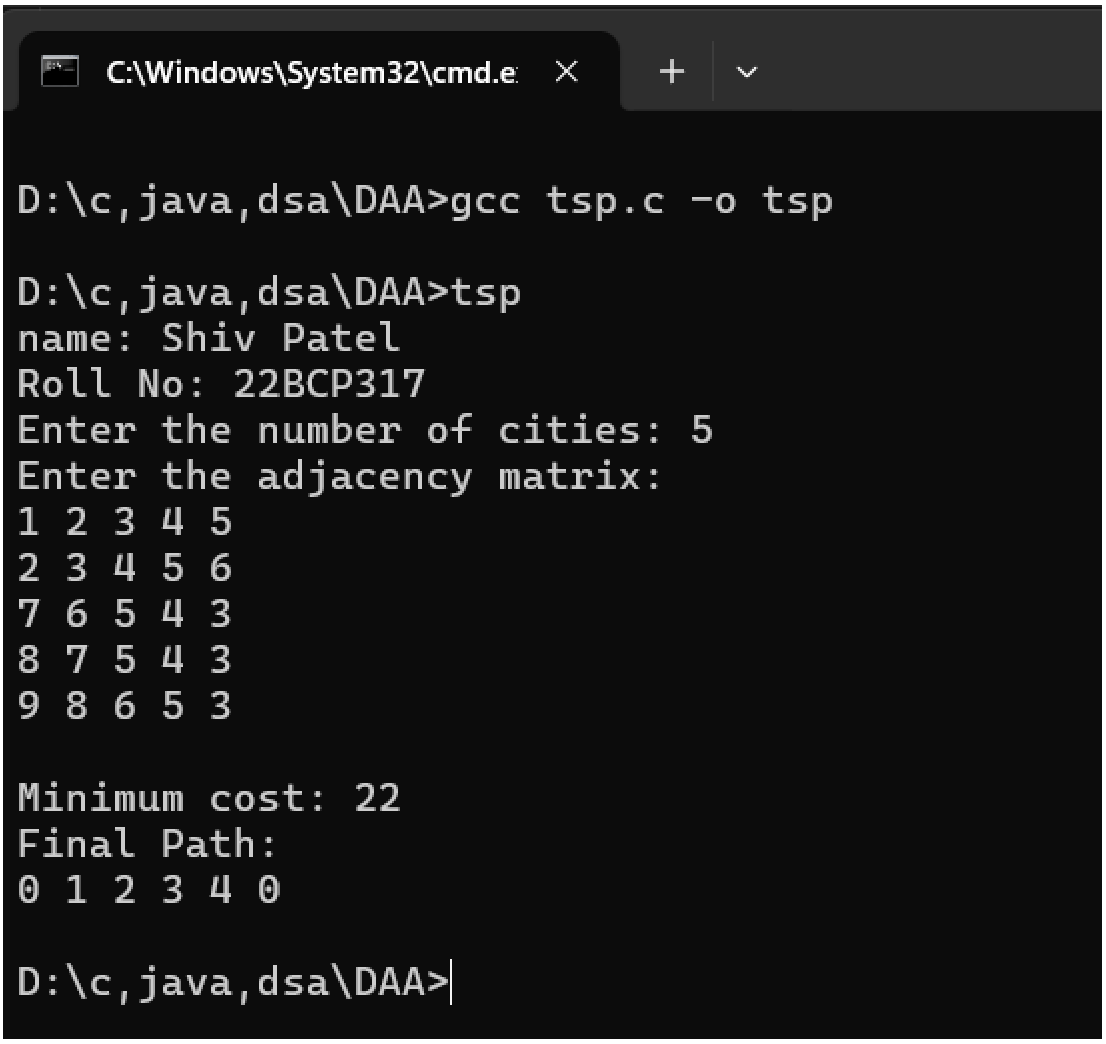
9	**[Branch and Bound]** Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point. Solve this problem using branch and bound technique. For example, consider the following graph:  A Travelling Salesman Problem (TSP) tour in the graph is $0 - 1 - 3 - 2 - 0$. The cost of the tour is $10 + 25 + 30 + 15 = 80$.

Source code:

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

#define N_MAX 10

int N;
int adjacency[N_MAX][N_MAX];
int finalPath[N_MAX + 1];
bool visited[N_MAX];
int minCost = __INT_MAX__;

void copyPathToFinal(int currentPath[])
{
   for (int i = 0; i < N; i++)
      finalPath[i] = currentPath[i];
   finalPath[N] = currentPath[0];
}

int firstMinimum(int i)
{
   int minimum = __INT_MAX__;
   for (int k = 0; k < N; k++)
      if (adjacency[i][k] < minimum && i != k)
         minimum = adjacency[i][k];
   return minimum;
}

int secondMinimum(int i)
{
   int first = __INT_MAX__, second = __INT_MAX__;
```

```
    for (int j = 0; j < N; j++)
    {
        if (i == j)
            continue;
        if (adjacency[i][j] <= first)
        {
            second = first;
            first = adjacency[i][j];
        }
        else if (adjacency[i][j] <= second && adjacency[i][j] != first)
        {
            second = adjacency[i][j];
        }
    }
    return second;
}

void tspRec(int currentBound, int currentWeight, int level, int currentPath[])
{

    if (level == N)
    {
        if (adjacency[currentPath[level - 1]][currentPath[0]] != 0)
        {
            int currentResult = currentWeight + adjacency[currentPath[level - 1]][currentPath[0]];
            if (currentResult < minCost)
            {
                copyPathToFinal(currentPath);
                minCost = currentResult;
            }
        }
        return;
    }

    for (int i = 0; i < N; i++)
    {

        if (adjacency[currentPath[level - 1]][i] != 0 && visited[i] == false)
        {
            int temp = currentBound;
            currentWeight += adjacency[currentPath[level - 1]][i];

            if (level == 1)
                currentBound -= ((firstMinimum(currentPath[level - 1]) + firstMinimum(i)) / 2);
            else
                currentBound -= ((secondMinimum(currentPath[level - 1]) + firstMinimum(i)) / 2);

            if (currentBound + currentWeight < minCost)
            {
                currentPath[level] = i;
                visited[i] = true;
                tspRec(currentBound, currentWeight, level + 1, currentPath);
            }
            currentWeight -= adjacency[currentPath[level - 1]][i];
            currentBound = temp;
            for (int j = 0; j <= level - 1; j++)
                visited[currentPath[j]] = true;
        }
    }
}

void tspSolver()
{
    int currentPath[N_MAX + 1];
    int currentBound = 0;

    for (int i = 0; i < N; i++)
        currentBound += (firstMinimum(i) + secondMinimum(i));
    currentBound = (currentBound == 1) ? currentBound / 2 + 1 : currentBound / 2;
    visited[0] = true;
    currentPath[0] = 0;
    tspRec(currentBound, 0, 1, currentPath);
}

int main()
{
    printf("name: Shiv Patel\n");
    printf("Roll No: 22BCP317\n");
    printf("Enter the number of cities: ");
    scanf("%d", &N);

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            scanf("%d", &adjacency[i][j]);
        }
    }

    tspSolver();

    printf("\nMinimum cost: %d\n", minCost);
    printf("Final Path:\n");
    for (int i = 0; i <= N; i++)
    {
        printf("%d ", finalPath[i]);
    }
    printf("\n");
    return 0;
}
```

Output:



```
D:\c,java,dsa\DAA>gcc tsp.c -o tsp

D:\c,java,dsa\DAA>tsp
name: Shiv Patel
Roll No: 22BCP317
Enter the number of cities: 5
Enter the adjacency matrix:
1 2 3 4 5
2 3 4 5 6
7 6 5 4 3
8 7 5 4 3
9 8 6 5 3

Minimum cost: 22
Final Path:
0 1 2 3 4 0

D:\c,java,dsa\DAA>
```

10	To design and solve given problems using different algorithmic approaches and analyze their complexity.
(I)	Your friends are starting a security company that needs to obtain licenses for n different pieces of cryptographic software. Due to regulations, they can only obtain these licenses at the rate of at most one per month. Each license is currently selling for a price of \$100. However, they are all becoming more expensive according to exponential growth curves: in particular, the cost of license j increases by a factor of $r_j > 1$ each month, where r_j is a given parameter. This means that if license j is purchased t months from now, it will cost $100r_jt$. We will assume that all the price growth rates are distinct; that is, $r_i \neq r_j$ for licenses $i \neq j$ (even though they start at the sameprice of \$100). The question is: Given that the company can only buy at most one license a month, in which order should it buy the licenses so that the total amount of money it spends is as small as possible? Give an algorithm that takes the n rates of price growth $r_1, r_2, \ldots, r_n$, and computes an order in which to buy the licenses so that the total amount ofmoney spent is minimized. The running time of your algorithm should be polynomial in n.
(II)	Suppose you are given an array A with n entries, with each entry holding a distinct number. You are told that the sequence of values $A[1], A[2], \ldots, A[n]$ is unimodal. That is, for some index p between 1 and n, the values in the array entries increase up to position p in A and then decrease the remainder of the way until position n. (So if you were to draw a plot with the array position j on the x-axis and the value of the entry $A[j]$ on the y-axis, the plotted points would rise until x-value p, where they'd achieve their maximum value, and then fall from there on). You'd like to find the "peak entry" p without having to read the entire array - in fact, by reading as few entries of A as possible. Show how to find the entry p by reading at most $O(logn)$ entries of A.

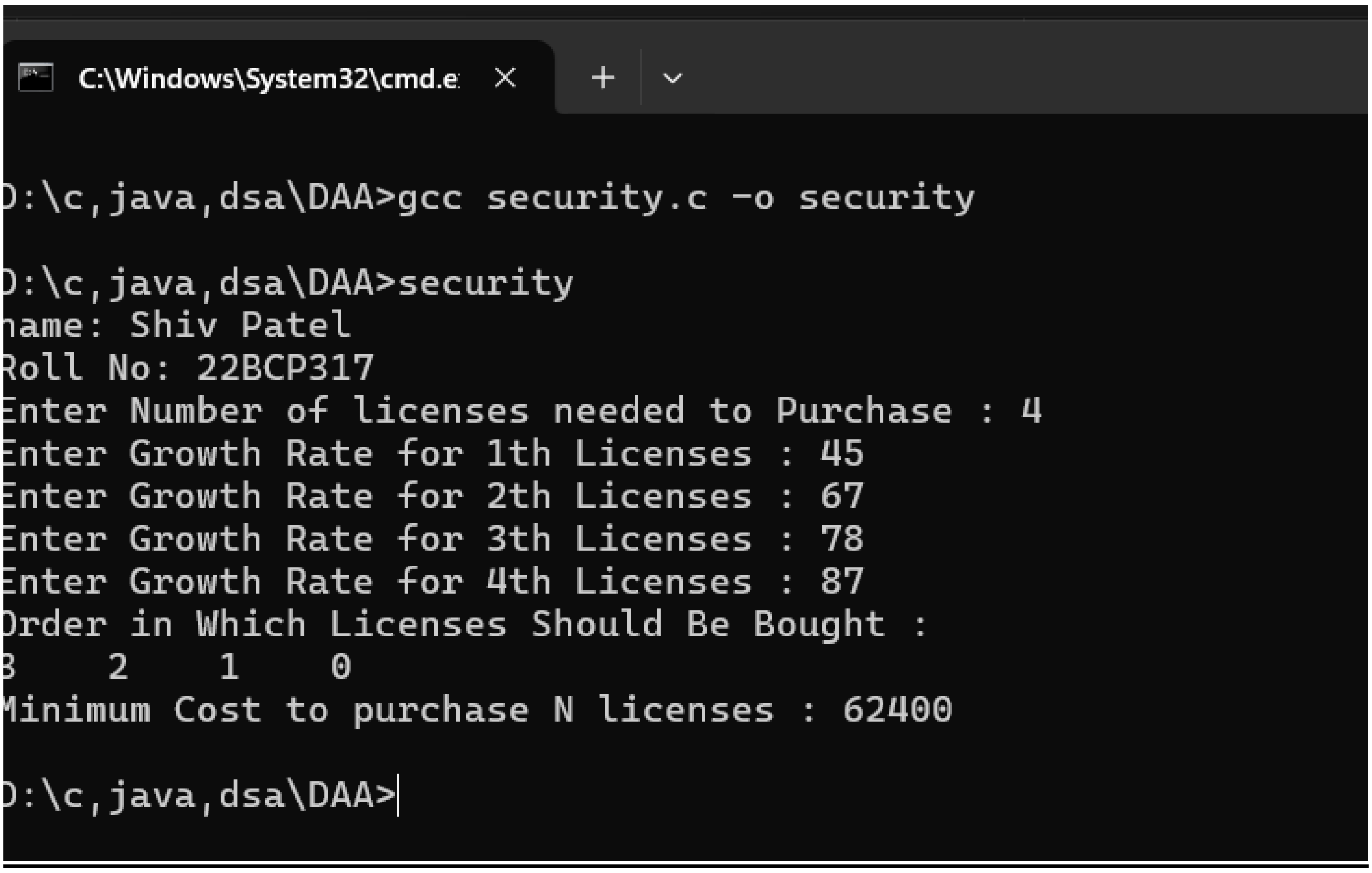
(Q1)

Source code:

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

struct Pair
{
   double first;
   int second;
};

int cmp(const struct Pair *a, const struct Pair *b)
{
   return (a->first < b->first) ? -1 : ((a->first > b->second) ? 1 : 0);
}

int main()
{
   printf("name: Shiv Patel\n");
   printf("Roll No: 22BCP317\n");
   printf("Enter Number of licenses needed to Purchase : ");
   int n;
```

```
    scanf("%d", &n);
    struct Pair rates[n];

    for (int i = 0; i < n; i++)
    {
        printf("Enter Growth Rate for %dth Licenses : ", i + 1);
        scanf("%lf", &rates[i].first);
        rates[i].second = i;
    }

    qsort(rates, n, sizeof(struct Pair), (int (*)(const void *, const void *))cmp);

    printf("Order in Which Licenses Should Be Bought :\n");
    int minCost = 0;
    for (int i = n - 1; i >= 0; i--)
    {
        printf("%d    ", rates[i].second);
        minCost = minCost + 100 * rates[i].first * (n - i);
    }

    printf("\nMinimum Cost to purchase N licenses : %d\n", minCost);

    return 0;
}
```

Output:



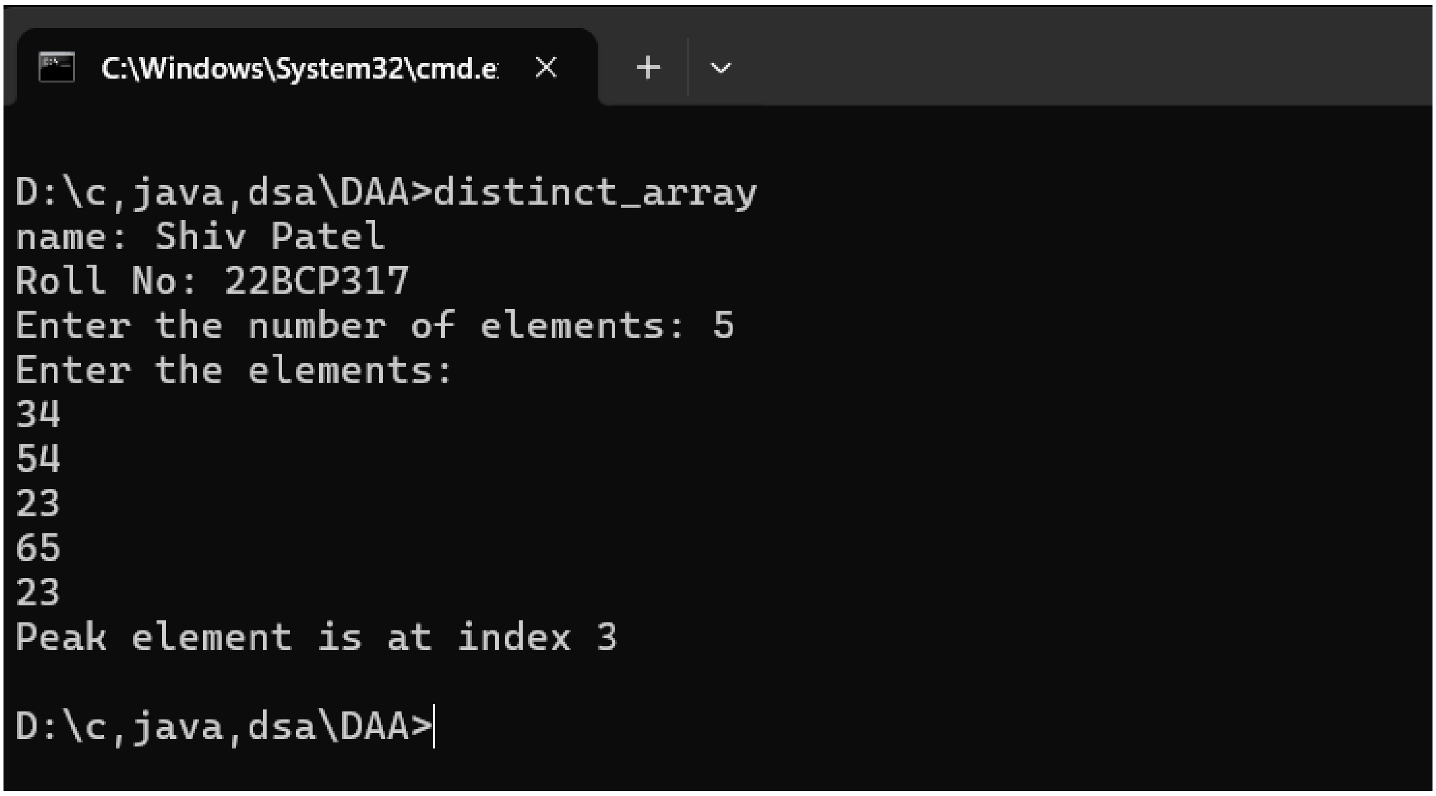
```
):\c,java,dsa\DAA>gcc security.c -o security

):\c,java,dsa\DAA>security
ıame: Shiv Patel
Roll No: 22BCP317
Enter Number of licenses needed to Purchase : 4
Enter Growth Rate for 1th Licenses : 45
Enter Growth Rate for 2th Licenses : 67
Enter Growth Rate for 3th Licenses : 78
Enter Growth Rate for 4th Licenses : 87
Order in Which Licenses Should Be Bought :
3    2    1    0
Minimum Cost to purchase N licenses : 62400

):\c,java,dsa\DAA>
```

(Q2)

Source code:

```
#include <stdio.h>

int findPeak(int arr[], int n)
{
   int left = 0;
   int right = n - 1;

   while (left < right)
   {
      int mid = left + (right - left) / 2;

      if (arr[mid] > arr[mid + 1])
      {
         right = mid;
      }
      else
      {
         left = mid + 1;
      }
   }

   return left;
}

int main()
{
   int n;
   printf("name: Shiv Patel\n");
   printf("Roll No: 22BCP317\n");
   printf("Enter the number of elements: ");
   scanf("%d", &n);
   int arr[n];
   printf("Enter the elements:\n");
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &arr[i]);
   }
   n = sizeof(arr) / sizeof(arr[0]);
   int peakIndex = findPeak(arr, n);

   printf("Peak element is at index %d\n", peakIndex);

   return 0;
}
```

Output:



```
D:\c,java,dsa\DAA>distinct_array
name: Shiv Patel
Roll No: 22BCP317
Enter the number of elements: 5
Enter the elements:
34
54
23
65
23
Peak element is at index 3

D:\c,java,dsa\DAA>
```