EXPERIMENT 1

AIM: To implement Linear Search.

PSEUDOCODE:

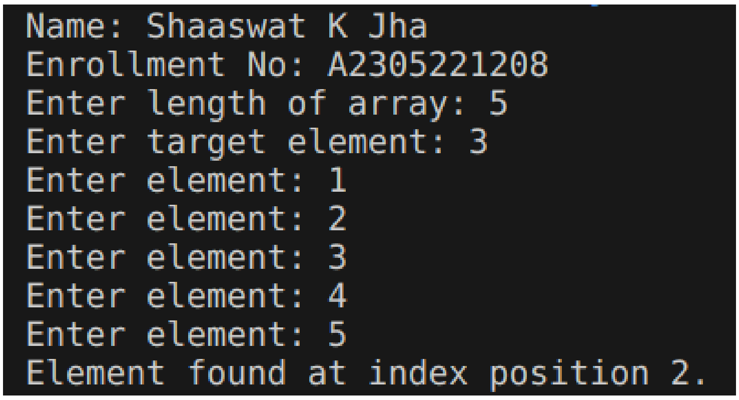
```
linearSearch(array(n), target)
    for i from 0 to n-1
        if array[i] = target
            return i
```

CODE:

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

int linearSearch(int array[], int target) {
    for (int i = 0; i < 10; i++)
    {
        if (array[i] == target)
        {
            return i;
        }
    }
    return -1;
}

int main()
{
    int length;
    int target;

    cout << "Enter length of array: ";
    cin >> length;
    cout << "Enter target element: ";
    cin >> target;

    int arr[length];

    for (int i=0; i < length; i++) {
        cout << "Enter element: ";
        cin >> arr[i];
    }

    int res = linearSearch(arr, target);

    if (res == -1)
    {
        cout << "Element not found!" << endl;
    } else {
        cout << "Element found at index position " << res << "." << endl;
    }
```

```
        return 0;
}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter length of array: 5
Enter target element: 3
Enter element: 1
Enter element: 2
Enter element: 3
Enter element: 4
Enter element: 5
Element found at index position 2.
```

TIME COMPLEXITY: O(n)

EXPERIMENT 2

AIM: To implement Binary Search:
(i) with Iterative Method (ii) with Recursive Method

PSEUDOCODE:

(i) With Iterative Method

```
binarySearch(array(n), target, low, high)
    do until low = high
        mid = (low + high) / 2
        if target = array[mid]
            return mid
        if target > array[mid]
            low = mid + 1
        else
            high = mid - 1
```

(ii) With Recursive Method

```
binarySearchWithRecursion(array(n), target, low, high)
    if low > high
        display “Element not found!”
    else
        mid = (low + high) / 2
        if target = array[mid]
            return mid
        if target > array[mid]
            return binarySearch(array(n) target, mid + 1, high)
        else
            return binarySearch(array(n), target, low, mid - 1)
```

CODE:

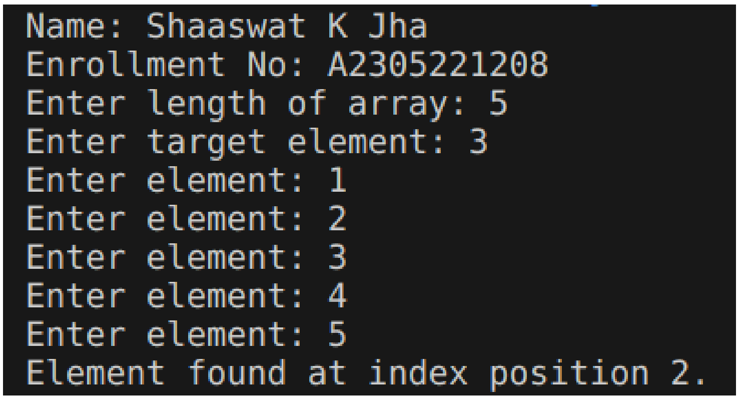
(i) With Iterative Method

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

int binarySearch(int array[], int x, int low, int high) {

    while (low <= high) {
        int mid = (low + high) / 2;

        if (array[mid] == x) {
            return mid;
        } else if (array[mid] < x) {
            low = mid + 1;
        } else {
            high = mid – 1;
        }
    }

    return -1;
}

int main()
{
    int length;
    int target;

    cout << "Enter length of array: ";
    cin >> length;
    cout << "Enter target element: ";
    cin >> target;

    int arr[length];

    for (int i=0; i < length; i++) {
        cout << "Enter element: ";
        cin >> arr[i];
    }

    int res = binarySearch(arr, target, 0, length);

    if (res == -1) {
        cout << "Element not found!" << endl;
    } else {
        cout << "Element found at index position " << res << "." << endl;
    }

    return 0;
}
```

(ii) With Recursive Method

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

int binarySearchWithRecursion(int array[], int x, int low, int high) {

    if (high >= low) {
        int mid = (low + high) / 2;

        if (array[mid] == x) {
            return mid;
        } else if (array[mid] > x) {
            return binarySearchWithRecursion(array, x, low, mid – 1);
        }
        return binarySearchWithRecursion(array, x, mid + 1, high);
    }
    return -1;
}

int main()
{
    int length;
    int target;


    cout << "Enter length of array: ";
    cin >> length;
    cout << "Enter target element: ";
    cin >> target;

    int arr[length];

    for (int i=0; i < length; i++) {
        cout << "Enter element: ";
        cin >> arr[i];
    }

    int res = binarySearchWithRecursion(arr, target, 0, length);

    if (res == -1)
    {
        cout << "Element not found!" << endl;
    } else {
        cout << "Element found at index position " << res << "." << endl;
    }
    return 0;
}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter length of array: 5
Enter target element: 3
Enter element: 1
Enter element: 2
Enter element: 3
Enter element: 4
Enter element: 5
Element found at index position 2.
```

TIME COMPLEXITY: O(logn)

EXPERIMENT 3

AIM: To implement Quicksort.

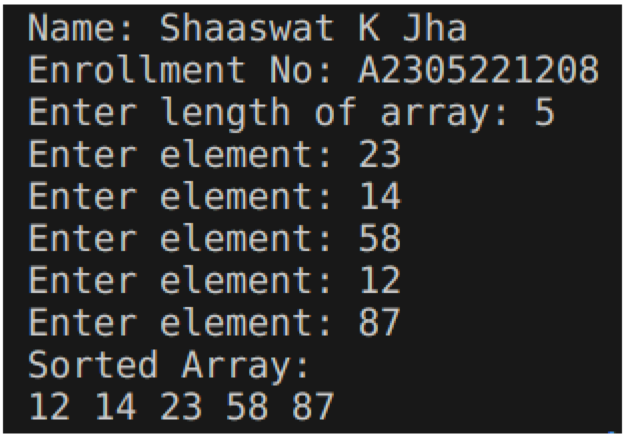
PSEUDOCODE:

```
quickSort(array, leftmostIndex, rightmostIndex)
  if (leftmostIndex < rightmostIndex)
    pivotIndex <- partition(array,leftmostIndex, rightmostIndex)
    quickSort(array, leftmostIndex, pivotIndex - 1)
    quickSort(array, pivotIndex, rightmostIndex)

partition(array, leftmostIndex, rightmostIndex)
  set rightmostIndex as pivotIndex
  storeIndex <- leftmostIndex - 1
  for i <- leftmostIndex + 1 to rightmostIndex
  if element[i] < pivotElement
    swap element[i] and element[storeIndex]
    storeIndex++
  swap pivotElement and element[storeIndex+1]
return storeIndex + 1
```

CODE:

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

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

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

int partition(int array[], int low, int high) {

 int pivot = array[high];
  int i = (low - 1);
 for (int j = low; j < high; j++) {
   if (array[j] <= pivot) {
     i++;

     swap(&array[i], &array[j]);
   }
 }
  swap(&array[i + 1], &array[high]);
  return (i + 1);
}

void quickSort(int array[], int low, int high) {
 if (low < high) {

   int pi = partition(array, low, high);

   quickSort(array, low, pi - 1);

   quickSort(array, pi + 1, high);
 }
}

int main()
{
   int length;
   int target;

   cout << "Enter length of array: ";
   cin >> length;

   int arr[length];

   for (int i=0; i < length; i++) {
       cout << "Enter element: ";
       cin >> arr[i];
   }

   quickSort(arr, 0, length - 1);
   cout << "Sorted Array:" << endl;

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

   cout << endl;

   return 0;
}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter length of array: 5
Enter element: 23
Enter element: 14
Enter element: 58
Enter element: 12
Enter element: 87
Sorted Array:
12 14 23 58 87
```

TIME COMPLEXITY:

Best case: Ω(nlogn)

Average Case: θ(nlogn)

Worst Case: O(n^2)

EXPERIMENT 4

AIM: To implement Mergesort.

PSEUDOCODE:

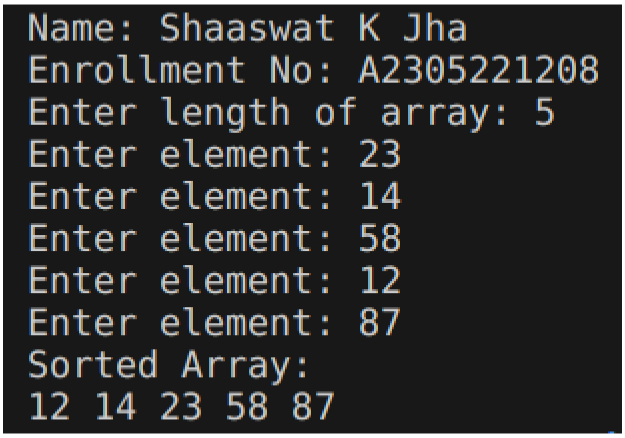
```
MergeSort(A, p, r):
    if p > r
        return
    q = (p+r)/2
    mergeSort(A, p, q)
    mergeSort(A, q+1, r)
    merge(A, p, q, r)
```

CODE:

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


void merge(int arr[], int p, int q, int r) {
  int n1 = q - p + 1;
 int n2 = r - q;


 int L[n1], M[n2];


 for (int i = 0; i < n1; i++) {
   L[i] = arr[p + i];
 }
 for (int j = 0; j < n2; j++) {
   M[j] = arr[q + 1 + j];
 }


 int i, j, k;
 i = 0;
 j = 0;
 k = p;


 while (i < n1 && j < n2) {
   if (L[i] <= M[j]) {
     arr[k] = L[i];
     i++;
   } else {
```

```
   arr[k] = M[j];
   j++;
  }
  k++;
 }

 while (i < n1) {
  arr[k] = L[i];
  i++;
  k++;
 }

 while (j < n2) {
  arr[k] = M[j];
  j++;
  k++;
 }
}

void mergeSort(int arr[], int l, int r) {
 if (l < r) {
  int m = l + (r - l) / 2;

  mergeSort(arr, l, m);
  mergeSort(arr, m + 1, r);
  merge(arr, l, m, r);
 }
}

int main()
{
  int length;
  int target;

  cout << "Enter length of array: ";
  cin >> length;

  int arr[length];
  for (int i=0; i < length; i++) {
    cout << "Enter element: ";
    cin >> arr[i];
  }

  mergeSort(arr, 0, length - 1);

  cout << "Sorted Array:" << endl;

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

  cout << endl;

  return 0;
}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter length of array: 5
Enter element: 23
Enter element: 14
Enter element: 58
Enter element: 12
Enter element: 87
Sorted Array:
12 14 23 58 87
```

TIME COMPLEXITY: O(nlogn)

EXPERIMENT 5

AIM:

To implement:
(i) Bubble Sort
(ii) Insertion Sort
(iii) Selection Sort

PSUEDOCODE:

(i) Bubble Sort

```
bubbleSort(array)
  for i <- 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap leftElement and rightElement
end bubbleSort
```

(ii) Insertion Sort

```
insertionSort(array)
  mark first element as sorted
  for each unsorted element X
    'extract' the element X
    for j <- lastSortedIndex down to 0
      if current element j > X
        move sorted element to the right by 1
    break loop and insert X here
end insertionSort
```

(iii) Selection Sort

```
selectionSort(array, size)
  repeat (size - 1) times
  set the first unsorted element as the minimum
  for each of the unsorted elements
    if element < currentMinimum
      set element as new minimum
  swap minimum with first unsorted position
end selectionSort
```

CODE:

(i) Bubble Sort

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

void bubbleSort(int array[], int length) {

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

   for (int j = 0; j < length - i; j++) {

     if (array[j] > array[j + 1]) {

       int temp = array[j];
       array[j] = array[j + 1];
       array[j + 1] = temp;
     }
   }
 }
}

int main()
```

```
{
    int length;
    int target;

    cout << "Enter length of array: ";
    cin >> length;

    int arr[length];

    for (int i=0; i < length; i++) {
        cout << "Enter element: ";
        cin >> arr[i];
    }

    bubbleSort(arr, length);

    cout << "Sorted Array:" << endl;

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

    cout << endl;

    return 0;
}
```

(ii) Insertion Sort

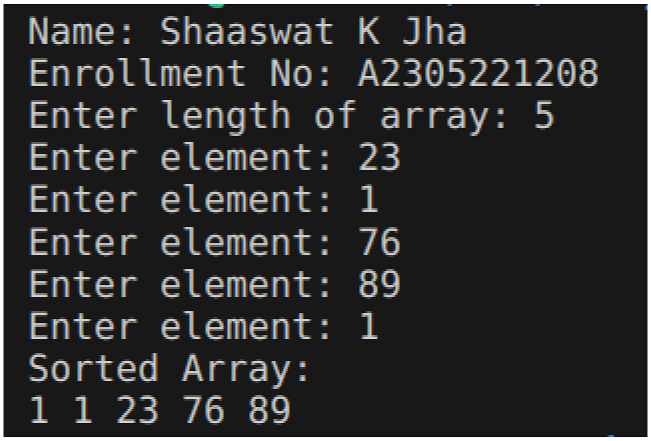
```
#include <iostream>
using namespace std;

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

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

int main()
{
   int length;
   int target;

   cout << "Enter length of array: ";
   cin >> length;

   int arr[length];

   for (int i=0; i < length; i++) {
       cout << "Enter element: ";
       cin >> arr[i];
   }

   insertionSort(arr, length);

   cout << "Sorted Array:" << endl;

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

   cout << endl;

   return 0;
}
```

(iii) Selection Sort

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

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

void selectionSort(int array[], int length) {
 for (int i = 0; i < length - 1; i++) {
   int min_idx = i;
   for (int j = i + 1; j < length; j++) {

     if (array[j] < array[min_idx])
       min_idx = j;
   }

   swap(&array[min_idx], &array[i]);
 }
}

int main()
{
   int length;
   int target;

   cout << "Enter length of array: ";
   cin >> length;

   int arr[length];

   for (int i=0; i < length; i++) {
       cout << "Enter element: ";
       cin >> arr[i];
   }

   selectionSort(arr, length);

   cout << "Sorted Array:" << endl;
```

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

  cout << endl;

  return 0;
}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter length of array: 5
Enter element: 23
Enter element: 1
Enter element: 76
Enter element: 89
Enter element: 1
Sorted Array:
1 1 23 76 89
```

TIME COMPLEXITY:

(i) Bubble Sort: $O(n^2)$
(ii) Insertion Sort: $O(n^2)$
(iii) Selection Sort: $O(n^2)$

EXPERIMENT 6

AIM:

To implement Fractional Knapsack Greedy Approach

PSUEDOCODE:

```
FractionalKnapsack(items[], n, capacity)
    Sort items[] based on decreasing value-to-weight ratio
    totalValue = 0
    for i from 0 to n-1
        if capacity == 0:
            break
        if items[i].weight <= capacity
            totalValue += items[i].value
            capacity -= items[i].weight
        else:
            totalValue += (items[i].value / items[i].weight) * capacity
            break
    return totalValue
```

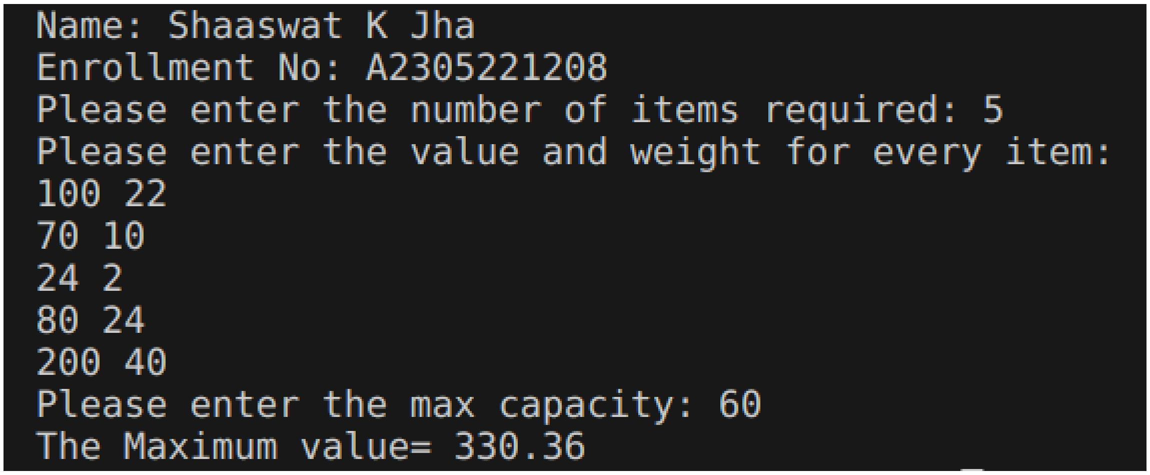
CODE:

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

void details() {
  cout << "Name: Shaaswat K Jha" << endl;
  cout << "Enrollment No: A2305221208" << endl;
}

struct Item
{
  int value;
  int weight;
};
```

```
int function(const void *a, const void *b)
{
    struct Item *itemA = (struct Item *) a;
    struct Item *itemB = (struct Item *) b;
    double ratioA = (double) itemA->value / itemA->weight;
    double ratioB = (double) itemB->value / itemB->weight;
    if (ratioA < ratioB)
        return 1;
    else if (ratioA > ratioB)
        return -1;
    else
        return 0;
}

double fractionalKnapsack(struct Item items[], int n, int capacity)
{
    qsort (items, n, sizeof (struct Item), function);
    double totalValue = 0.0;
    for (int i = 0; i < n; ++i)
    {
        if (capacity == 0)
            break;
        if (items[i].weight <= capacity)
        {
            totalValue += items[i].value;
            capacity -= items[i].weight;
        }
        else
        {
            totalValue += ((double) items[i].value / items[i].weight) * capacity;
            break;
        }
```

```
  }
  return totalValue;
}

int main()
{
  details();

  int n;
  printf ("Please enter the number of items required: ");
  scanf ("%d", &n);
  struct Item items[n];
  printf ("Please enter the value and weight for every item:\n");
  for (int i = 0; i < n; ++i)
  {
     scanf ("%d %d", &items[i].value, &items[i].weight);
  }
  int capacity;
  printf ("Please enter the max capacity: ");
  scanf ("%d", &capacity);
  double maxValue = fractionalKnapsack (items, n, capacity);
  printf ("The Maximum value= %.2lf\n", maxValue);
  return 0;
}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Please enter the number of items required: 5
Please enter the value and weight for every item:
100 22
70 10
24 2
80 24
200 40
Please enter the max capacity: 60
The Maximum value= 330.36
```

TIME COMPLEXITY: O(nlogn)

EXPERIMENT 7

AIM:

To implement:
(i) Prim’s Algorithm
(ii) Kruskal’s Algorithm

PSEUDOCODE:

```
Input: Graph G represented by an adjacency matrix or adjacency list
1. Initialize an empty set MST to store the Minimum Spanning Tree.
2. Select a starting vertex startVertex.
3. Create an empty priority queue pq.
4. Insert (startVertex, 0) into pq, where 0 is the initial key value.
5. While pq is not empty:
    a. Extract the vertex u with the minimum key value from pq.
    b. Add u to the MST set.
    c. For each vertex v adjacent to u:
    - If v is not in MST and the edge weight u-v is smaller than the current key
    value of v:
        i. Update the key value of vertex v in pq to the edge weight u-v.
        ii. Update the parent of vertex v to be u.
6. Once all vertices are processed, the MST set will contain the Minimum
Spanning Tree.
```

CODE:

(i) Prim’s Algorithm

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

```
void details() {
  cout << "Name: Shaaswat K Jha" << endl;
  cout << "Enrollment No: A2305221208" << endl;
}

const int MAX_VERTICES = 20;

int findMinKey(int key[], bool mstSet[], int V) {
  int min = INT_MAX, min_index;
  for (int v = 0; v < V; v++) {
    if (!mstSet[v] && key[v] < min) {
      min = key[v];
      min_index = v;
    }
  }
  return min_index;
}

void primMST(int graph[MAX_VERTICES][MAX_VERTICES], int V) {
  int parent[MAX_VERTICES];
  int key[MAX_VERTICES];
  bool mstSet[MAX_VERTICES];

  for (int i = 0; i < V; i++) {
    key[i] = INT_MAX;
    mstSet[i] = false;
  }
  key[0] = 0;
  parent[0] = -1;
```

```
    for (int count = 0; count < V - 1; count++) {
        int u = findMinKey(key, mstSet, V);
        mstSet[u] = true;
        for (int v = 0; v < V; v++) {
            if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }
    cout << "Edge\tWeight" << endl;
    for (int i = 1; i < V; i++) {
        cout << parent[i] << " - " << i << "\t" << graph[i][parent[i]] << endl;
    }
}

int main() {
    int V;
    details();
    cout << "Enter the number of vertices: ";
    cin >> V;
    int graph[MAX_VERTICES][MAX_VERTICES];
    cout << "Enter the adjacency matrix:" << endl;
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            cin >> graph[i][j];
        }
    }
    primMST(graph, V);
```

```
  return 0;
}
```

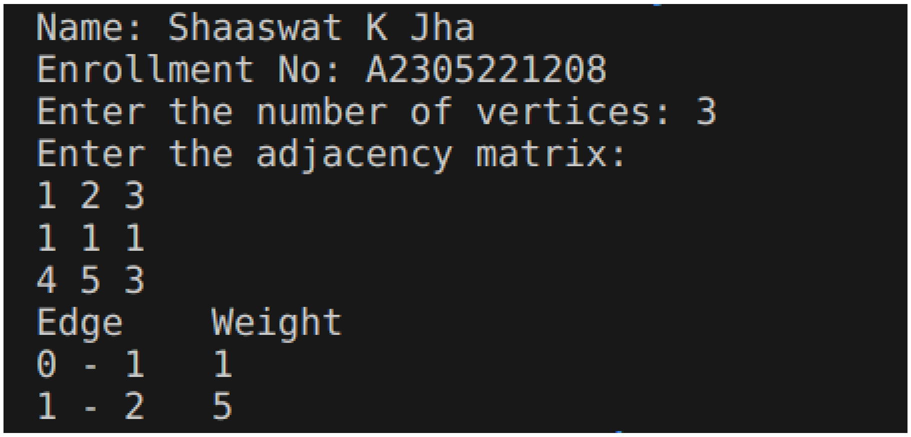
(ii) Kruskal’s Algorithm

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

void details() {
  cout << "Name: Shaaswat K Jha" << endl;
  cout << "Enrollment No: A2305221208" << endl;
}

const int MAX_VERTICES = 20;
const int MAX_EDGES = 50;

struct Edge {
  int src, dest, weight;
};

struct Graph {
  int V, E;
  Edge edges[MAX_EDGES];
};

int findParent(vector<int>& parent, int i) {
  if (parent[i] == -1)
     return i;
```

```
    return findParent(parent, parent[i]);
}

void unionSets(vector<int>& parent, int x, int y) {
    int xroot = findParent(parent, x);
    int yroot = findParent(parent, y);
    parent[xroot] = yroot;
}

bool compareEdges(const Edge& a, const Edge& b) {
    return a.weight < b.weight;
}

void kruskalMST(Graph* graph) {
    int V = graph->V;
    Edge result[MAX_VERTICES];
    int e = 0;
    int i = 0;

    sort(graph->edges, graph->edges + graph->E, compareEdges);

    vector<int> parent(V, -1);

    while (e < V - 1 && i < graph->E) {
        Edge next_edge = graph->edges[i++];
        int x = findParent(parent, next_edge.src);
        int y = findParent(parent, next_edge.dest);
        if (x != y) {
            result[e++] = next_edge;
            unionSets(parent, x, y);
        }
    }
    cout << "Edge\tWeight" << endl;
    for (i = 0; i < e; i++) {
        cout << result[i].src << " - " << result[i].dest << "\t" << result[i].weight <<
endl;
    }
}

int main() {
    Graph graph;

    details();

    cout << "Enter the number of vertices: ";
    cin >> graph.V;
    cout << "Enter the number of edges: ";
    cin >> graph.E;

    cout << "Enter edge details (src, dest, weight):" << endl;
    for (int i = 0; i < graph.E; i++) {
        cin >> graph.edges[i].src >> graph.edges[i].dest >> graph.edges[i].weight;
    }

    kruskalMST(&graph);

    return 0;
}
```

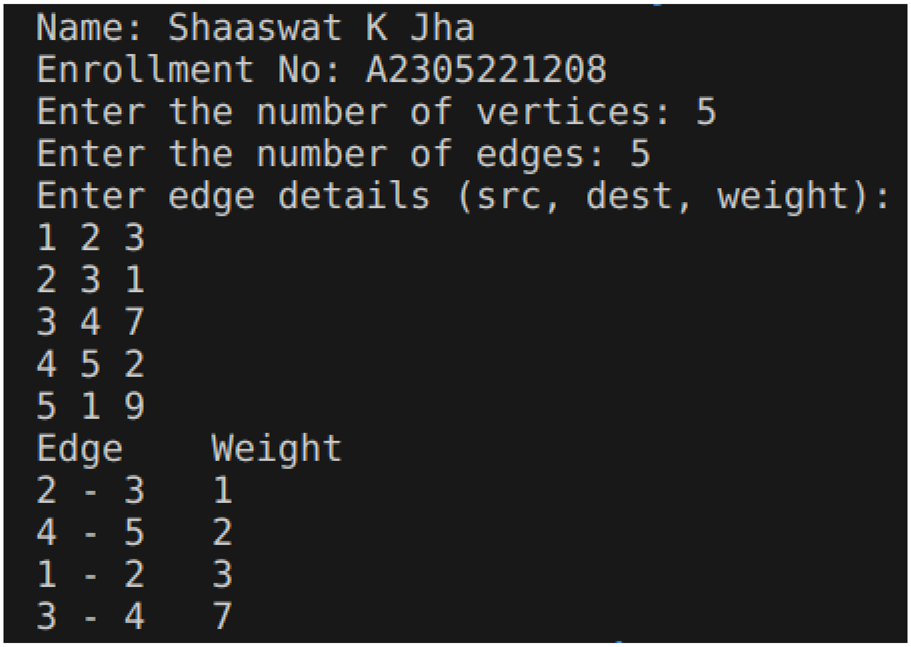
OUTPUT:

(i) Prim’s Algorithm



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter the number of vertices: 3
Enter the adjacency matrix:
1 2 3
1 1 1
4 5 3
Edge    Weight
0 - 1   1
1 - 2   5
```

(ii) Kruskal’s Algorithm



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter the number of vertices: 5
Enter the number of edges: 5
Enter edge details (src, dest, weight):
1 2 3
2 3 1
3 4 7
4 5 2
5 1 9
Edge    Weight
2 - 3   1
4 - 5   2
1 - 2   3
3 - 4   7
```

EXPERIMENT 8

AIM:

To implement Dijkstra’s Algorithm

PSEUDOCODE:

```
function dijkstra(G, S)
    for each vertex V in G
        distance[V] <- infinite
        previous[V] <- NULL
        If V != S, add V to Priority Queue Q
    distance[S] <- 0

    while Q IS NOT EMPTY
        U <- Extract MIN from Q
        for each unvisited neighbour V of U
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] <- tempDistance
                previous[V] <- U
    return distance[], previous[]
```

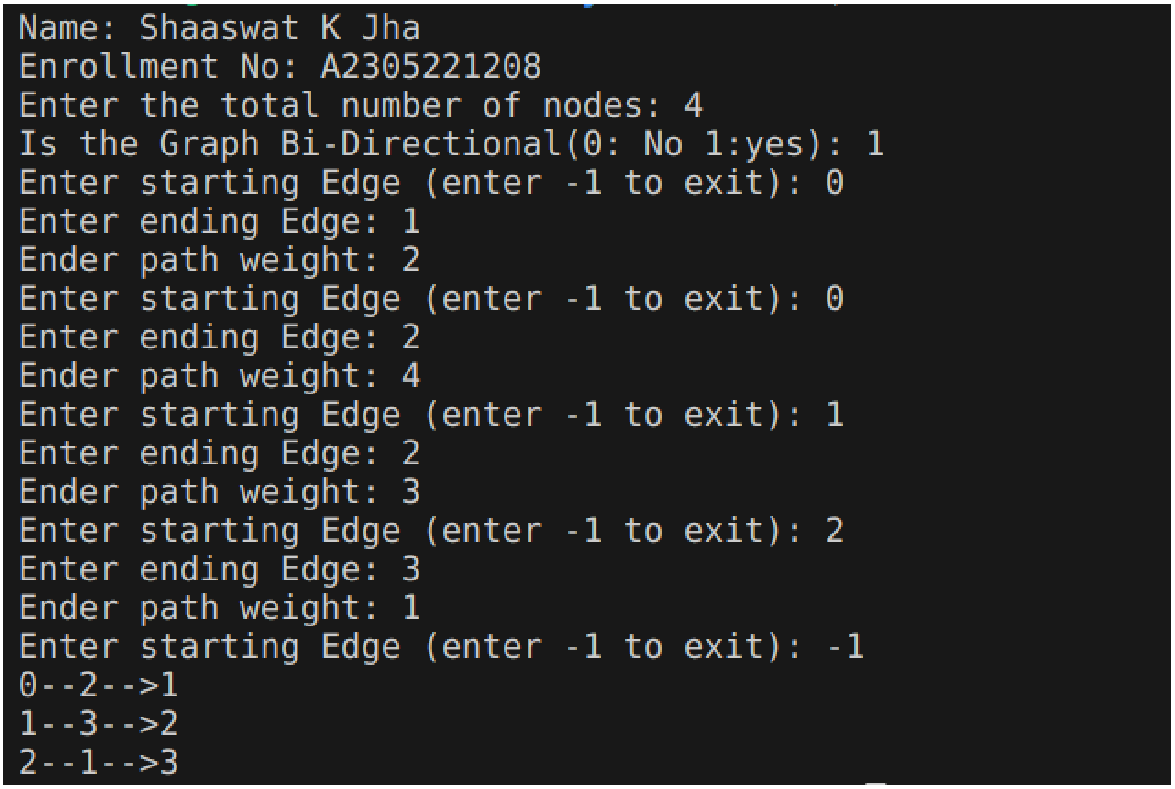
CODE:

```
#include <iostream>
#include <limits.h>
#include <vector>
#include <algorithm>

int inf = INT_MAX;
```

```
using namespace std;

void details() {
  cout << "Name: Shaaswat K Jha" << endl;
  cout << "Enrollment No: A2305221208" << endl;
}

int ** Graph(int nodes){
  int** Graph = new int*[nodes];
  for(int i = 0;i<nodes;i++){
    Graph[i] = new int[nodes];
    for(int j = 0;j<nodes;j++){
      if(i==j){
        Graph[i][j]=0;
      }
      else{
        Graph[i][j]= inf;
      }
    }
  }
  return(Graph);
}
int ** create_graph(int ** Graph,int nodes){
  int bi_directional;
  cout<<"Is the Graph Bi-Directional(0: No 1:yes): ";
  cin>>bi_directional;

  while(true){
    int i,j;
```

```
        cout<<"Enter starting Edge (enter -1 to exit): ";
        cin>>i;
        if(i==-1){
            break;
        }
        cout<<"Enter ending Edge: ";
        cin>>j;
        int w;
        cout<<"Ender path weight: ";
        cin>>w;

        Graph[i][j] = w;
        if(bi_directional==1){
            Graph[j][i] = w;
        }
    }
    return Graph;
}
void print_graph(int ** graph,int nodes){
    cout <<" \t";
    for(int i =0;i<nodes;i++){
        cout<<i<<"\t";
    }
    cout<<endl;
    for(int i =0;i<nodes;i++){
        cout<<i<<"\t";
        for(int j=0;j<nodes;j++){
            if(graph[i][j]==inf){
                cout<<"inf"<<"\t";
            }
            else{
                cout<<graph[i][j]<<"\t";
            }
        }
        cout<<endl;
    }
}
int node_in(vector<int> v,int j){
    for(auto i = v.begin();i<v.end();i++){
        if(j==*i){
            return(1);
        }
    }
    return(0);
}
int main(){

    details();

    int nodes;
    cout<<"Enter the total number of nodes: ";
    cin>>nodes;
    int ** graph = Graph(nodes);
    graph = create_graph(graph,nodes);

    vector<int> nodes_covered;
    nodes_covered.push_back(0);
    while(nodes_covered.size()<nodes){
        int min_weight = -1;
        int min_end_node = -1;
        int min_start_node = -1;
        for(auto i = nodes_covered.begin();i<nodes_covered.end();i++)
        {
            for(int j = 0;j<nodes;j++){
                if(node_in(nodes_covered,j)==0){
                    if(min_weight==-1 || min_weight>graph[*i][j]){
                        min_weight = graph[*i][j];
                        min_end_node = j;
                        min_start_node = *i;
                    }
                }
            }
        }
        cout<<min_start_node<<"--"<<min_weight<<"-->"<<min_end_node<<endl;
        nodes_covered.push_back(min_end_node);

    }

    return 0;
}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter the total number of nodes: 4
Is the Graph Bi-Directional(0: No 1:yes): 1
Enter starting Edge (enter -1 to exit): 0
Enter ending Edge: 1
Ender path weight: 2
Enter starting Edge (enter -1 to exit): 0
Enter ending Edge: 2
Ender path weight: 4
Enter starting Edge (enter -1 to exit): 1
Enter ending Edge: 2
Ender path weight: 3
Enter starting Edge (enter -1 to exit): 2
Enter ending Edge: 3
Ender path weight: 1
Enter starting Edge (enter -1 to exit): -1
0--2-->1
1--3-->2
2--1-->3
```

PRACTICAL 9

AIM:

To implement the Strassen Matrix Multiplication.

PSEUDOCODE:

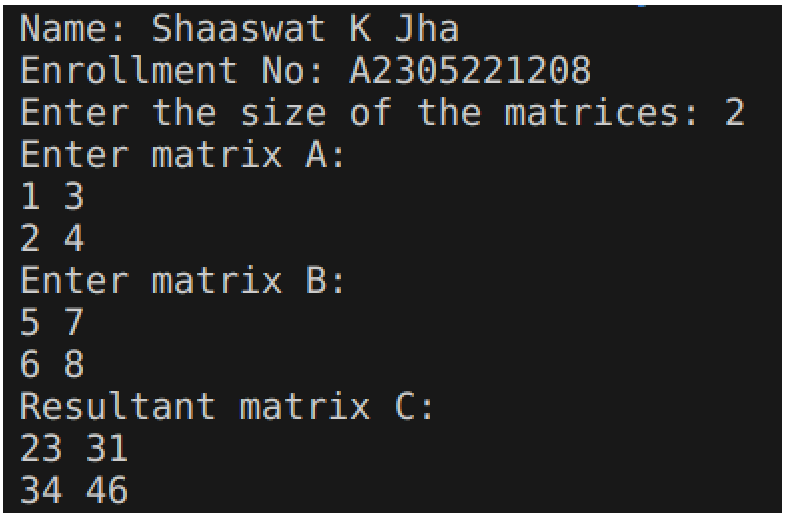
```
function strassen(A, B):
    if size(A) == 1:
        return A * B
    // Divide the matrices into submatrices
    A11, A12, A21, A22 = split(A)
    B11, B12, B21, B22 = split(B)
    // Calculate the intermediate matrices
    M1 = strassen(A11 + A22, B11 + B22)
    M2 = strassen(A21 + A22, B11)
    M3 = strassen(A11, B12 - B22)
    M4 = strassen(A22, B21 - B11)
    M5 = strassen(A11 + A12, B22)
    M6 = strassen(A21 - A11, B11 + B12)
    M7 = strassen(A12 - A22, B21 + B22)
    // Calculate the result submatrices
    C11 = M1 + M4 - M5 + M7
    C12 = M3 + M5
    C21 = M2 + M4
    C22 = M1 - M2 + M3 + M6
    // Combine the result submatrices
    C = join(C11, C12, C21, C22)
    return C
```

CODE:

```
#include <iostream>
#include <vector>

using namespace std;

vector<vector<int>> matrixAddition(const vector<vector<int>>& A, const
vector<vector<int>>& B) {
    int n = A.size();
    vector<vector<int>> result(n, vector<int>(n, 0));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = A[i][j] + B[i][j];
        }
    }

    return result;
}

vector<vector<int>> matrixSubtraction(const vector<vector<int>>& A, const
vector<vector<int>>& B) {
    int n = A.size();
    vector<vector<int>> result(n, vector<int>(n, 0));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = A[i][j] - B[i][j];
        }
    }

    return result;
}

vector<vector<int>> strassenMultiply(const vector<vector<int>>& A, const
vector<vector<int>>& B) {
    int n = A.size();

    if (n == 1) {
        vector<vector<int>> result(1, vector<int>(1, 0));
        result[0][0] = A[0][0] * B[0][0];
        return result;
    }

    int halfN = n / 2;

    vector<vector<int>> A11(halfN, vector<int>(halfN));
    vector<vector<int>> A12(halfN, vector<int>(halfN));
    vector<vector<int>> A21(halfN, vector<int>(halfN));
    vector<vector<int>> A22(halfN, vector<int>(halfN));

    vector<vector<int>> B11(halfN, vector<int>(halfN));
    vector<vector<int>> B12(halfN, vector<int>(halfN));
    vector<vector<int>> B21(halfN, vector<int>(halfN));
    vector<vector<int>> B22(halfN, vector<int>(halfN));

    for (int i = 0; i < halfN; i++) {
        for (int j = 0; j < halfN; j++) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + halfN];
```

```
      A21[i][j] = A[i + halfN][j];
      A22[i][j] = A[i + halfN][j + halfN];

      B11[i][j] = B[i][j];
      B12[i][j] = B[i][j + halfN];
      B21[i][j] = B[i + halfN][j];
      B22[i][j] = B[i + halfN][j + halfN];
    }
  }

  vector<vector<int>> P1 = strassenMultiply(A11, matrixSubtraction(B12,
B22));
  vector<vector<int>> P2 = strassenMultiply(matrixAddition(A11, A12), B22);
  vector<vector<int>> P3 = strassenMultiply(matrixAddition(A21, A22), B11);
  vector<vector<int>> P4 = strassenMultiply(A22, matrixSubtraction(B21,
B11));
  vector<vector<int>> P5 = strassenMultiply(matrixAddition(A11, A22),
matrixAddition(B11, B22));
  vector<vector<int>> P6 = strassenMultiply(matrixSubtraction(A12, A22),
matrixAddition(B21, B22));
  vector<vector<int>> P7 = strassenMultiply(matrixSubtraction(A11, A21),
matrixAddition(B11, B12));

  vector<vector<int>> C11 =
matrixSubtraction(matrixAddition(matrixAddition(P5, P4), P6), P2);
  vector<vector<int>> C12 = matrixAddition(P1, P2);
  vector<vector<int>> C21 = matrixAddition(P3, P4);
  vector<vector<int>> C22 =
matrixSubtraction(matrixSubtraction(matrixAddition(P5, P1), P3), P7);

  vector<vector<int>> result(n, vector<int>(n, 0));
  for (int i = 0; i < halfN; i++) {
        for (int j = 0; j < halfN; j++) {
            result[i][j] = C11[i][j];
            result[i][j + halfN] = C12[i][j];
            result[i + halfN][j] = C21[i][j];
            result[i + halfN][j + halfN] = C22[i][j];
        }
    }

    return result;
}

int main() {
    int n;
    cout << "Enter the size of the matrices: ";
    cin >> n;

    vector<vector<int>> A(n, vector<int>(n));
    vector<vector<int>> B(n, vector<int>(n));

    cout << "Enter matrix A:" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> A[i][j];
        }
    }

    cout << "Enter matrix B:" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> B[i][j];
        }
    }
    vector<vector<int>> result = strassenMultiply(A, B);

    cout << "Resultant matrix C:" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << result[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter the size of the matrices: 2
Enter matrix A:
1 3
2 4
Enter matrix B:
5 7
6 8
Resultant matrix C:
23 31
34 46
```

TIME COMPLEXITY: O($n^{2.81}$)

PRACTICAL 10

AIM:

To solve the longest common subsequence problem.

PSEUDOCODE:

```
X and Y be two given sequences
Initialize a table LCS of dimension X.length * Y.length
X.label = X
Y.label = Y
LCS[0][] = 0
LCS[][0] = 0
Start from LCS[1][1]
Compare X[i] and Y[j]
   If X[i] = Y[j]
      LCS[i][j] = 1 + LCS[i-1, j-1]
      Point an arrow to LCS[i][j]
   Else
      LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])
      Point an arrow to max(LCS[i-1][j], LCS[i][j-1])
```

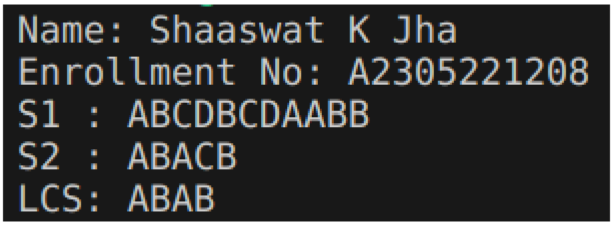
CODE:

```
#include <iostream>

#include <cstring>

using namespace std;
```

```
void lcsAlgo(char *S1, char *S2, int m, int n) {
 int LCS_table[m + 1][n + 1];

 for (int i = 0; i <= m; i++) {
  for (int j = 0; j <= n; j++) {
   if (i == 0 || j == 0)
    LCS_table[i][j] = 0;
   else if (S1[i - 1] == S2[j - 1])
    LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
   else
    LCS_table[i][j] = max(LCS_table[i - 1][j], LCS_table[i][j - 1]);
  }
 }

 int index = LCS_table[m][n];
 char lcsAlgo[index + 1];
 lcsAlgo[index] = '\0';

 int i = m, j = n;
 while (i > 0 && j > 0) {
  if (S1[i - 1] == S2[j - 1]) {
   lcsAlgo[index - 1] = S1[i - 1];
```

```
    i--;

    j--;

    index--;

   }


   else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])

    i--;

   else

    j--;

 }

  cout << "S1 : " << S1 << "\nS2 : " << S2 << "\nLCS: " << lcsAlgo << "\n";

}

int main() {

 char S1[] = "ABCDBCDAABB";

 char S2[] = "ABACB";

 int m = strlen(S1);

 int n = strlen(S2);

 lcsAlgo(S1, S2, m, n);

}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
S1 : ABCDBCDAABB
S2 : ABACB
LCS: ABAB
```

PRACTICAL 11

AIM:

To solve 0/1 Knapsack Problem using Dynamic Programming.

PSEUDOCODE:

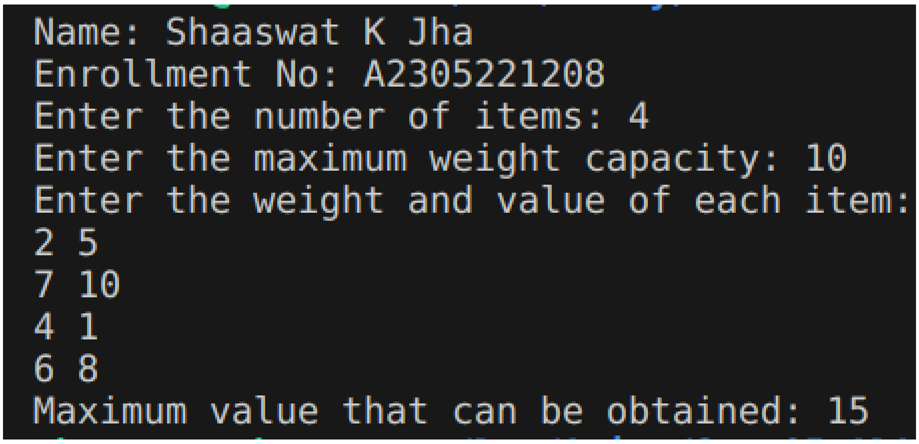
```
function knapsackDP(items, W):
  n = length(items)
  create a 2D array dp of size (n+1) x (W+1)
  for i from 0 to n:
      for w from 0 to W:
          if i is 0 or w is 0:
              dp[i][w] = 0
          else if weight of items[i-1] > w:
              dp[i][w] = dp[i-1][w]
          else:
              dp[i][w] = max(dp[i-1][w], value of items[i-1] + dp[i-1][w - weight of items[i-1]])
  return dp[n][W]
items = an array of items with weights and values
W = maximum weight capacity of the knapsack
result = knapsackDP(items, W)
```

CODE:

```
#include <iostream>
#include <vector>

using namespace std;

struct Item {
	int weight;
	int value;
};
```

```
}

int knapsack(vector<Item>& items, int W) {
    int n = items.size();
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for (int i = 1; i <= n; i++) {
    for (int w = 1; w <= W; w++) {
    if (items[i - 1].weight > w) {
        dp[i][w] = dp[i - 1][w];
    } else {
        dp[i][w] = max(dp[i - 1][w], items[i - 1].value + dp[i - 1][w - items[i - 1].weight]);
    }
    }
    }

    return dp[n][W];
}

int main() {
    int n, W;
    cout << "Enter the number of items: ";
    cin >> n;
    cout << "Enter the maximum weight capacity: ";
    cin >> W;

    vector<Item> items(n);

    cout << "Enter the weight and value of each item:" << endl;
    for (int i = 0; i < n; i++) {
    cin >> items[i].weight >> items[i].value;
    }

    int maxValue = knapsack(items, W);

    cout << "Maximum value that can be obtained: " << maxValue << endl;

    return 0;
}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter the number of items: 4
Enter the maximum weight capacity: 10
Enter the weight and value of each item:
2 5
7 10
4 1
6 8
Maximum value that can be obtained: 15
```

PRACTICAL 12

AIM:

To implement Breadth First Search.

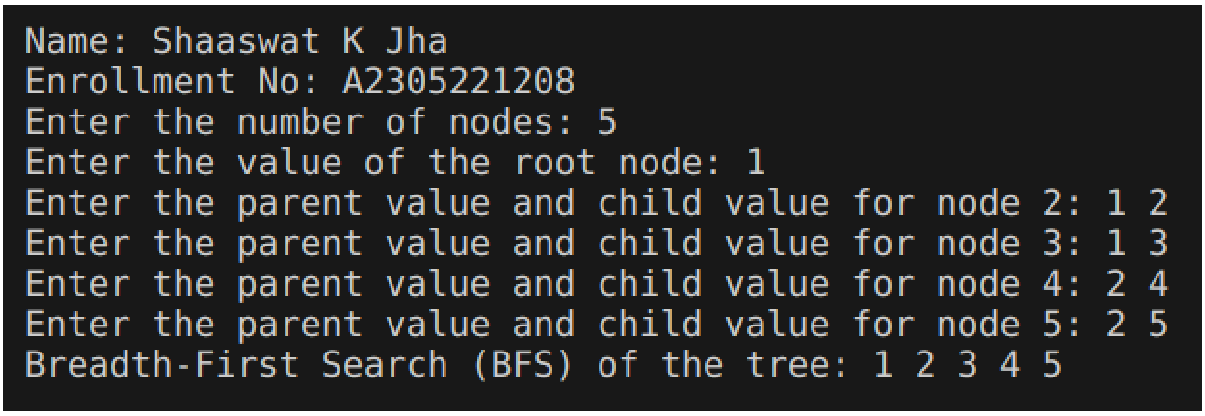
PSEUDOCODE:

```
BFS(G, startVertex):
        Initialize an empty queue Q
        Initialize a boolean array visited with a size of |V| (the number of vertices)
        Initialize all elements of visited to false
        Q.enqueue(startVertex)
        visited[startVertex] = true
        while (Q is not empty):
        currentVertex = Q.dequeue()
        Process(currentVertex)
        for each adjacentVertex in G.adjacencyList[currentVertex]:
                if (visited[adjacentVertex] is false):
                Q.enqueue(adjacentVertex)
                visited[adjacentVertex] = true
  BFS(graph, startVertex)
```

CODE:

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

struct TreeNode {
  int data;
  vector<TreeNode*> children;
  TreeNode(int value) : data(value) {}
};

void BFS(TreeNode* root) {
    if (!root)
        return;

    queue<TreeNode*> q;
    q.push(root);

    while (!q.empty()) {
        TreeNode* current = q.front();
        cout << current->data << " ";
        q.pop();

        for (TreeNode* child : current->children) {
            if (child)
                q.push(child);
        }
    }
}

TreeNode* buildTree() {
    int n, rootValue;
    cout << "Enter the number of nodes: ";
    cin >> n;

    if (n <= 0) {
        cout << "Invalid input. Please enter a positive number of nodes." << endl;
        return nullptr;
    }

    cout << "Enter the value of the root node: ";
    cin >> rootValue;

    TreeNode* root = new TreeNode(rootValue);
    vector<TreeNode*> nodeQueue;
    nodeQueue.push_back(root);

    for (int i = 1; i < n; i++) {
        int parentValue, childValue;
        cout << "Enter the parent value and child value for node " << i + 1 << ": ";
        cin >> parentValue >> childValue;
        TreeNode* newNode = new TreeNode(childValue);
        for (TreeNode* node : nodeQueue) {
            if (node->data == parentValue) {
                node->children.push_back(newNode);
                nodeQueue.push_back(newNode);
                break;
            }
        }
    }

    return root;
}
int main() {
    TreeNode* root = buildTree();
    if (root) {
        cout << "Breadth-First Search (BFS) of the tree: ";
        BFS(root);
        cout << endl;
    }
     return 0;
}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter the number of nodes: 5
Enter the value of the root node: 1
Enter the parent value and child value for node 2: 1 2
Enter the parent value and child value for node 3: 1 3
Enter the parent value and child value for node 4: 2 4
Enter the parent value and child value for node 5: 2 5
Breadth-First Search (BFS) of the tree: 1 2 3 4 5
```

PRACTICAL 13

AIM:

To implement Depth First Search.

PSEUDOCODE:

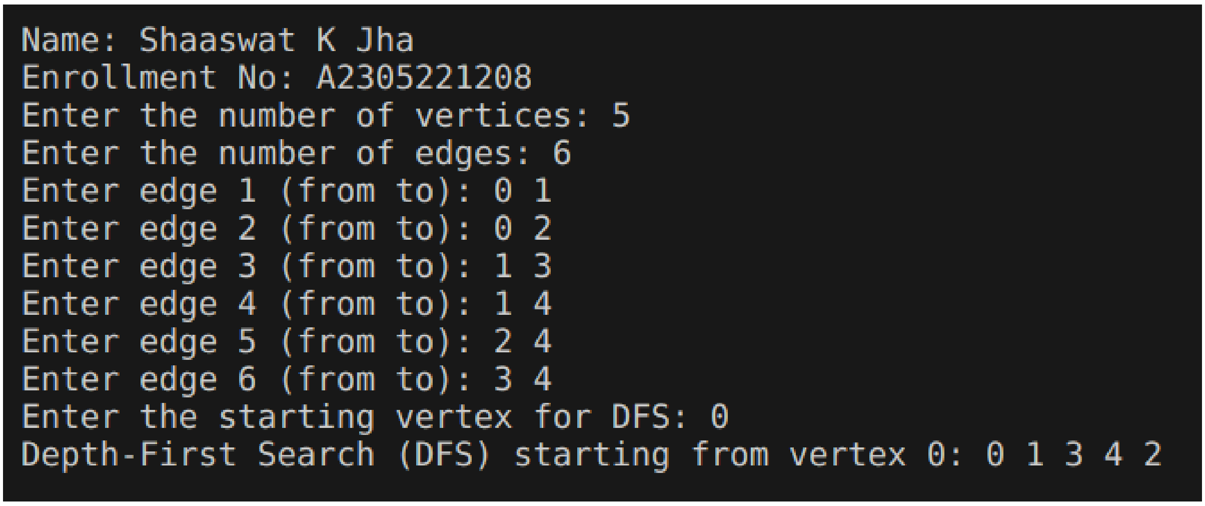
```
DFS(G, currentVertex, visited):
    visited[currentVertex] = true
    Process(currentVertex)
    for each adjacentVertex in G.adjacencyList[currentVertex]:
    if (visited[adjacentVertex] is false):
        DFS(G, adjacentVertex, visited)
Initialize a boolean array visited with a size of |V| (the number of vertices)
Initialize all elements of visited to false
for each vertex in G.vertices:
    if (visited[vertex] is false):
    DFS(G, vertex, visited)
DFS(graph, startVertex, visited)
```

CODE:

```
#include <iostream>
#include <vector>

using namespace std;

class Graph {
public:
        Graph(int vertices);
        void addEdge(int from, int to);
        void DFS(int startVertex);

private:
        int vertices;
        vector<vector<int>> adjacencyList;
        vector<bool> visited;
```

```
};

Graph::Graph(int vertices) {
    this->vertices = vertices;
    adjacencyList.resize(vertices);
    visited.resize(vertices, false);
}

void Graph::addEdge(int from, int to) {
    adjacencyList[from].push_back(to);
}

void Graph::DFS(int startVertex) {
    visited[startVertex] = true;
    cout << startVertex << " ";

    for (int adjacent : adjacencyList[startVertex]) {
    if (!visited[adjacent]) {
    DFS(adjacent);
    }
    }
}

int main() {
    int numVertices;
    cout << "Enter the number of vertices: ";
    cin >> numVertices;

    Graph g(numVertices);

    int numEdges;
    cout << "Enter the number of edges: ";
    cin >> numEdges;

    for (int i = 0; i < numEdges; i++) {
    int from, to;
    cout << "Enter edge " << i + 1 << " (from to): ";
    cin >> from >> to;
    g.addEdge(from, to);
    }

    int startVertex;
    cout << "Enter the starting vertex for DFS: ";
    cin >> startVertex;

    cout << "Depth-First Search (DFS) starting from vertex " << startVertex << ": ";
    g.DFS(startVertex);
    cout << endl;

    return 0;
}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter the number of vertices: 5
Enter the number of edges: 6
Enter edge 1 (from to): 0 1
Enter edge 2 (from to): 0 2
Enter edge 3 (from to): 1 3
Enter edge 4 (from to): 1 4
Enter edge 5 (from to): 2 4
Enter edge 6 (from to): 3 4
Enter the starting vertex for DFS: 0
Depth-First Search (DFS) starting from vertex 0: 0 1 3 4 2
```

PRACTICAL 14

AIM:

To solve N-Queen Problem.

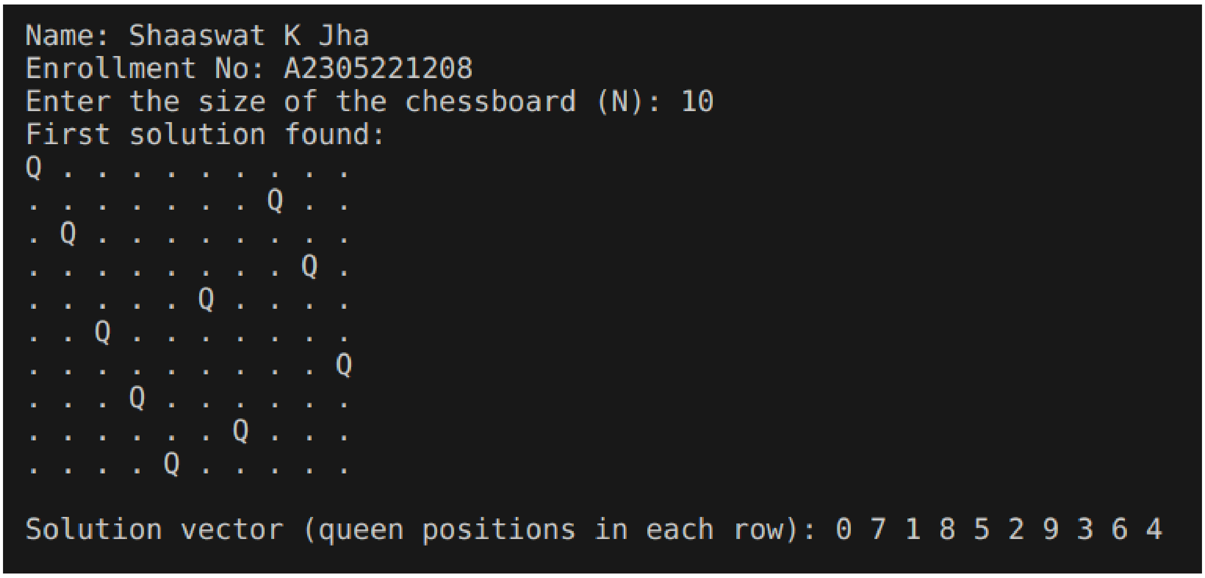
PSEUDOCODE:

```
NQueens(N):
    Create an empty N×N chessboard
    if PlaceQueens(N, 0, chessboard):
    Print the solution
    else:
    Print "No solution exists"
PlaceQueens(N, row, chessboard):
    if row == N:
    return true
    for each column in [0, N-1]:
    if IsSafe(row, column, chessboard):
        chessboard[row][column] = 'Q'
        if PlaceQueens(N, row + 1, chessboard):
        return true
        chessboard[row][column] = '.'
    return false
IsSafe(row, col, chessboard):
    for c in [0, col-1]:
    if chessboard[row][c] == 'Q':
        return false
    for r, c in [(row-1, col-1), (row-2, col-2), ..., (0, 0)]:
    if r < 0 or c < 0:
        break
    if chessboard[r][c] == 'Q':
        return false
    for r, c in [(row+1, col-1), (row+2, col-2), ..., (N-1, 0)]:
    if r >= N or c < 0:
        break
    if chessboard[r][c] == 'Q':
        return false
    return true
```

CODE:

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

void printChessboard(const vector<vector<char>>& chessboard) {
    for (const vector<char>& row : chessboard) {
    for (char cell : row) {
    cout << cell << " ";
    }
    cout << endl;
    }
    cout << endl;
}

bool isSafe(int row, int col, const vector<vector<char>>& chessboard, int N) {
    for (int c = 0; c < col; ++c) {
    if (chessboard[row][c] == 'Q') {
    return false;
    }
    }

    for (int r = row, c = col; r >= 0 && c >= 0; --r, --c) {
    if (chessboard[r][c] == 'Q') {
    return false;
    }
    }

    // Check the lower-left diagonal
    for (int r = row, c = col; r < N && c >= 0; ++r, --c) {
    if (chessboard[r][c] == 'Q') {
    return false;
    }
    }

    return true;
}
bool findFirstSolution(int col, int N, vector<vector<char>>& chessboard, vector<int>& solution) {
    if (col == N) {
    solution = vector<int>(N, -1);
    for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
        if (chessboard[i][j] == 'Q') {
        solution[i] = j;
        break;
        }
    }
    }
    return true;
    }

    for (int row = 0; row < N; ++row) {
    if (isSafe(row, col, chessboard, N)) {
    chessboard[row][col] = 'Q';
    if (findFirstSolution(col + 1, N, chessboard, solution)) {
        return true;
    }
    chessboard[row][col] = '.';
    }
    }

    return false;
}

int main() {
    int N;
    cout << "Enter the size of the chessboard (N): ";
    cin >> N;

    vector<vector<char>> chessboard(N, vector<char>(N, '.'));
    vector<int> solution;
    if (findFirstSolution(0, N, chessboard, solution)) {
    cout << "First solution found:" << endl;
    printChessboard(chessboard);
    cout << "Solution vector (queen positions in each row): ";
    for (int i = 0; i < N; ++i) {
    cout << solution[i] << " ";
    }
    cout << endl;
    } else {
    cout << "No solution found." << endl;
    }

    return 0;
}
```

OUTPUT:



```
Name: Shaaswat K Jha
Enrollment No: A2305221208
Enter the size of the chessboard (N): 10
First solution found:
Q . . . . . . . . .
. . . . . . . Q . .
. Q . . . . . . . .
. . . . . . . . Q .
. . . . . Q . . . .
. . Q . . . . . . .
. . . . . . . . . Q
. . . Q . . . . . .
. . . . . . Q . . .
. . . . Q . . . . .

Solution vector (queen positions in each row): 0 7 1 8 5 2 9 3 6 4
```