

**A.D.A – LAB**

**ETCS-351**

|  |  |
| --- | --- |
| **Submitted To:** | **Submitted By:** |
| **Ms. Gurpreet Kaur** | **Shivam Singh** |
| CSE Department | 07313302720 |
| A.D.A Lab | CSE-5A –(Group 2) |

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.NO** | **Name of Experiment** | **Date of Experiment** | **Date of Checking** | **Remarks** |
| **1.** | Write a program for linear search and binary search |  |  |  |
| **2.** | Write a program for insertion sort and selection sort |  |  |  |
| **3.** | Write a program for bubble sort and Radix sort |  |  |  |
| **4.** | Write a program for Merge sort and quick sort |  |  |  |
| **5.** | Write a program for Strassen’s Matrix Multiplication |  |  |  |
| **6.** | Write a program for Minimum Spanning Tree Algorithms – Prim’s & Kruskal’s Algorithm |  |  |  |
| **7.** | Write a program for Dijkstra’s Algorithm and Floyd Warshall Algorithm |  |  |  |
| **8.** | Write a program to implement 0-1 knapsack Problem |  |  |  |
| **9.** | Write a program to implement LCS |  |  |  |
| **10.** | Write a program Knuth-Morris-Pratt Algorithm |  |  |  |

**Basic Function**

The function used in this code is used throughout the file. This page can be used as a reference to later code used

**Code 🡺**

#include <string>

#include <iostream>

#define endl '\n'

using namespace std;

void printinfo() {

    cout << "Shivam Singh\n07313302720\nCSE-5A\n";

}

void printArray(int arr[], int size) {

    cout << "The entered array is: ";

    for(int i = 0; i < size ; i++) {

        cout << arr[i] << ' ';

    }

    cout << endl;

}

int\* takeInputArray (int size) {

    int \*tempArray = new int[size+1];

    cout << "Enter the elements of array separated by space: ";

    for(int i = 0 ; i < size ; i++) {

        cin >> tempArray[i];

    }

    return tempArray;

}

int sizeInput() {

    int size{};

    cout << "Enter the size of the array: ";

    cin >> size;

    return size;

}

void keyFound(int result) {

    if(result == -1) {

        cout << "Key not found\n";

    } else {

        cout << "Key found at index: " << result << '\n';

    }

}

int keyInput() {

    int key{};

    cout << "Enter the element you want to find: ";

    cin >> key;

    return key;

}

void swap(int\* xp, int\* yp)

{

    int temp = \*xp;

    \*xp = \*yp;

    \*yp = temp;

}

int getMax(int array[], int n) {

  int max = array[0];

  for (int i = 1; i < n; i++)

    if (array[i] > max)

      max = array[i];

  return max;

}

**Experiment-1**

**Aim 🡺 Write a program for Linear Search and Binary Search**

**Software Used 🡺 VScode, GNU’s C++ compiler**

**Code 🡺**

**Linear Search**

int ls(int \*arr, int size, int key) {

    int index{-1};

    for(int i = 0 ; i < size ; i++) {

        if(arr[i] == key) {

            index = i;

        }

    }

    return index;

}

**Binary Search**

#include <algorithm>

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

  std::sort(array, array+(high+1));

  if (high >= low) {

    int mid = low + (high - low) / 2;

    // If found at mid, then return it

    if (array[mid] == x)

      return mid;

    // Search the left half

    if (array[mid] > x)

      return bs(array, x, low, mid - 1);

    // Search the right half

    return bs(array, x, mid + 1, high);

  }

  return -1;

}

**Driver Code**

#include <iostream>

#include <string>

#include "linearSearch.h"

#include "binarySearch.h"

#include "../basicFunction.h"

using namespace std;

int main() {

    printinfo();

    //---------------------------- Taking User Input-----------------------------------------------------//

    int size = sizeInput();

    int key = keyInput();

    int \*array = new int[size+1];

    array = takeInputArray(size);

    printArray(array, size);

    cout << '\n';

    //-----------------------------Linear Search--------------------------------------------------------//

    cout << "Result of Linear Search\n";

    int resultLS{ls(array, size, key)};

    printArray(array, size);

    keyFound(resultLS);

    cout << "\n";

    //-----------------------------Binary Search-------------------------------------------------------//

    cout << "Result of Binary Search\n";

    int resultBS{bs(array,key,0,size-1)};

    printArray(array,size);

    keyFound(resultBS);

    cout << "\n";

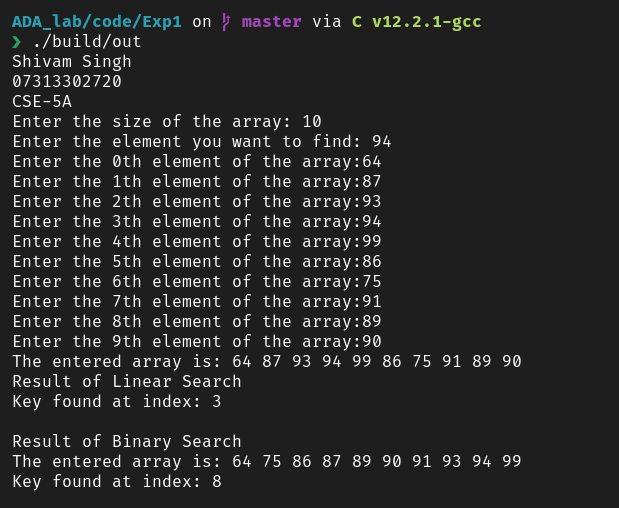
    //---------------------------------End of Code-----------------------------------------------------//

}

**Result 🡺**

Linear Search and Binary Search was implemented using C++

**Output**



**Experiment-2**

**Aim 🡺 To write a program to implement Insertion and Selection sort**

**Software Used 🡺 VScode, GNU’s C++ compiler**

**Code 🡺**

**Insertion sort**

void insertSort(int array[], int size) {

  for (int step = 1; step < size; step++) {

    int key = array[step];

    int j = step – 1;

    while (key < array[j] && j >= 0) {

      array[j + 1] = array[j];

      --j;

    }

    array[j + 1] = key;

  }

}

**Selection Sort**

void swap(int \*a, int \*b) {

  int temp = \*a;

  \*a = \*b;

  \*b = temp;

}

void selectSort(int array[], int size) {

  for (int step = 0; step < size – 1; step++) {

    int min\_idx = step;

    for (int i = step + 1; i < size; i++) {

      if (array[i] < array[min\_idx])

        min\_idx = i;

    }

    swap(&array[min\_idx], &array[step]);

  }

}

**Main Code**

#include <iostream>

#include “../basicFunction.h”

#include “insertionSort.h”

#include “selectionSort.h”

using namespace std;

int main() {

    printinfo();

    //--------------------------------------Insertion Sort-----------------------------------------------------------//

    int size = sizeInput();

    int \*array = new int[size+1];

    array = takeInputArray(size);

    printArray(array, size);

    insertSort(array, size);

    cout << “(Insertion Sort)Sorted Array is: “;

    printArray(array,size);

    //-------------------------------------selection Sort------------------------------------------------------------//

    int size1 = sizeInput();

    int \*array1 = new int[size1+1];

    array1 = takeInputArray(size1);

    printArray(array1,size1);

    selectSort(array1,size1);

    cout << “(Selection Sort)Sorted Array is: “;

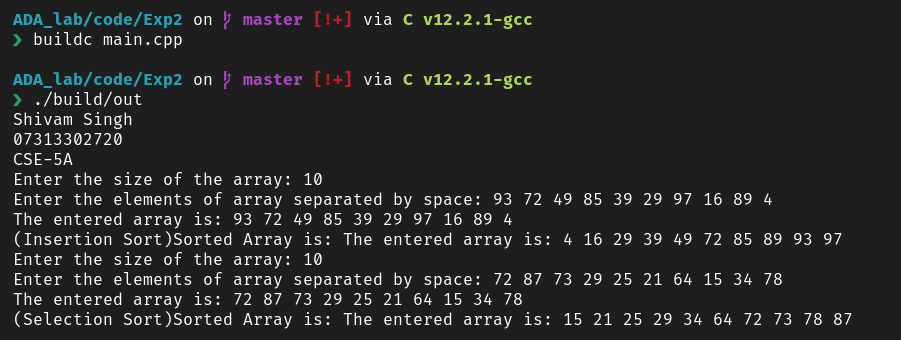
    printArray(array1, size1);

    return 0;

}

**Result 🡺**

Selection sort and insertion was implemented using C++

**Output**

**Experiment-3**

**Aim 🡺 Write a program for Bubble Sort and Radix Sort**

**Software Used 🡺 VScode, GNU’s C++ compiler**

**Code 🡺**

**//Bubble Sort**

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

  for (int i = 0; i < size-1; i++) {

    for (int j = 0; j < size-i-1; j++) {

      if (array[j] > array[j + 1]) {

        swap(&array[j],&array[j+1]);

      }

    }

  }

}

**//Radix Sort**

void countingSort(int array[], int size, int place) {

  const int max = 10;

  int output[size];

  int count[max];

  for (int i = 0; i < max; ++i)

    count[i] = 0;

  for (int i = 0; i < size; i++)

    count[(array[i] / place) % 10]++;

  for (int i = 1; i < max; i++)

    count[i] += count[i - 1];

  for (int i = size - 1; i >= 0; i--) {

    output[count[(array[i] / place) % 10] - 1] = array[i];

    count[(array[i] / place) % 10]--;

  }

  for (int i = 0; i < size; i++)

    array[i] = output[i];

}

void radixsort(int array[], int size) {

  int max = getMax(array, size);

  for (int place = 1; max / place > 0; place \*= 10)

    countingSort(array, size, place);

}

**//Main Driver Code**

#include <iostream>

#include "../basicFunction.h"

#include "bubbleSort.h"

#include "radixSort.h"

using namespace std;

int main() {

    printinfo();

    //--------------------------------------Bubble Sort--------------------------------------------------//

    int size{sizeInput()};

    int \*array = new int[size+1];

    array = takeInputArray(size);

    printArray(array,size);

    bubbleSort(array,size);

    cout << "(Bubble Sort)Sorted Array is: ";

    printArray(array,size);

    //-------------------------------------Radix Sort---------------------------------------------------//

    int size1{sizeInput()};

    int \*array1 = new int[size1+1];

    array1 = takeInputArray(size1);

    printArray(array1,size1);

    radixsort(array1,size1);

    cout << "(Radix Sort)Sorted Array is: ";

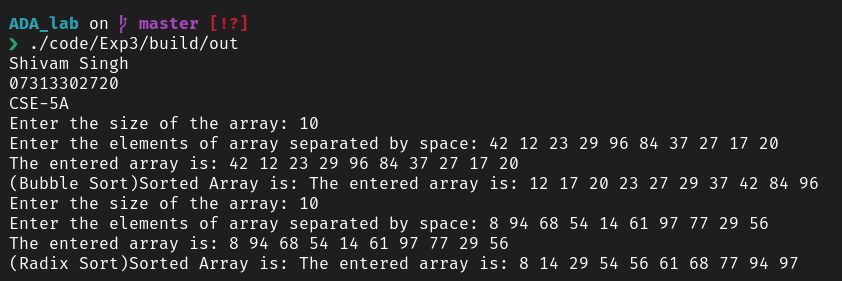
    printArray(array1,size1);

    return 0;

}

**Result 🡺** Bubble Sort and Radix Sort was implemented using C++

**Output**



**Experiment – 4**

**Aim 🡺 Write a program for Merge Sort and Quick Sort**

**Software Used 🡺 VSCode, GNU’s C++ compiler**

**Code 🡺**

**//**Merge Sort

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);

  }

}

//Quick Sort

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);

  }

}

//Driver Code

#include <iostream>

#include "../basicFunction.h"

#include "mergeSort.h"

#include "quickSort.h"

using namespace std;

int main() {

    printinfo();

    //------------------------------------------Merge Sort----------------------------------------//

     int size{sizeInput()};

    int \*array = new int[size+1];

    array = takeInputArray(size);

    printArray(array,size);

    mergeSort(array,0,size-1);

    cout << "(Merge Sort)Sorted Array is: ";

    printArray(array,size);

    //-----------------------------------------Quick Sort-----------------------------------------//

    int size1{sizeInput()};

    int \*array1 = new int[size1+1];

    array1 = takeInputArray(size1);

    printArray(array1,size1);

    quickSort(array1,0,size1-1);

    cout << "(Quick Sort)Sorted Array is: ";

    printArray(array1,size1);

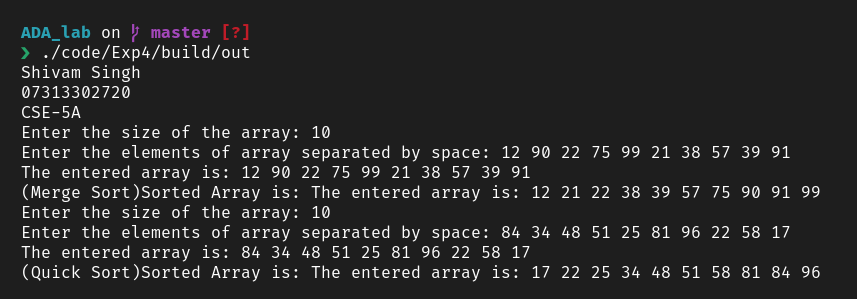
    return 0;

}

**Result 🡺**

Merge Sort and Quick Sort was implemented using C++

**//Output**

****

**Experiment – 5**

**Aim 🡺 Write a program for Strassen’s Matrix Multiplication**

**Software Used 🡺 VSCode, GNU’s C++ compiler**

**Code 🡺**

//Strassen’s Matrix Multiplication

#include <vector>

#define ROW\_1 4

#define COL\_1 4

#define ROW\_2 4

#define COL\_2 4

using namespace std;

void printMatrix(vector<vector<int> > matrix) {

    for(int i = 0; i < (int)matrix.size(); i++){

        for(int j = 0; j < (int)matrix[i].size(); j++){

            cout << matrix[i][j] << ' ';

        }

        cout << '\n';

    }

}

vector<vector<int>> add(vector<vector<int> > A, vector<vector<int> > B, int split\_index, int multiplier = 1) {

    for (auto i = 0; i < split\_index; i++)

        for (auto j = 0; j < split\_index; j++)

            A[i][j] = A[i][j] + (multiplier \* B[i][j]);

    return A;

}

vector<vector<int> > strassen\_multiplication(vector<vector<int> > A, vector<vector<int> > B) {

    // calculating the size of matrix

    int col\_1 = A[0].size();

    int row\_1 = A.size();

    int col\_2 = B[0].size();

    int row\_2 = B.size();

    // checking if multiplication is possible or not

    // between the input matrices

    if (col\_1 != row\_2) {

        cout << "The Two Matrices cannot be multiplied";

        return {};

    }

    // creating an empty matrix to store the result

    vector<int> result\_row(col\_2, 0);

    vector<vector<int> > result(row\_1, result\_row);

    // Base case

    // if size of matrix is 1

    if (col\_1 == 1)

        result[0][0]

            = A[0][0] \* B[0][0];

    else {

        // split index

        int split\_index = col\_1 / 2;

        vector<int> row\_vector(split\_index, 0);

        // Splitting the matrices in sub matrices

        vector<vector<int> > a00(split\_index, row\_vector);

        vector<vector<int> > a01(split\_index, row\_vector);

        vector<vector<int> > a10(split\_index, row\_vector);

        vector<vector<int> > a11(split\_index, row\_vector);

        vector<vector<int> > b00(split\_index, row\_vector);

        vector<vector<int> > b01(split\_index, row\_vector);

        vector<vector<int> > b10(split\_index, row\_vector);

        vector<vector<int> > b11(split\_index, row\_vector);

        // calculating and storing the result

        // inside our quadrants

        for (auto i = 0; i < split\_index; i++)

            for (auto j = 0; j < split\_index; j++) {

                a00[i][j] = A[i][j];

                a01[i][j] = A[i][j + split\_index];

                a10[i][j] = A[split\_index + i][j];

                a11[i][j] = A[i + split\_index]

                                    [j + split\_index];

                b00[i][j] = B[i][j];

                b01[i][j] = B[i][j + split\_index];

                b10[i][j] = B[split\_index + i][j];

                b11[i][j] = B[i + split\_index]

                                    [j + split\_index];

            }

        // Calculating the multiplication using the formula

        // given by strassent algorithm

        vector<vector<int>> p1(

            strassen\_multiplication(a00, add(b01, b11, split\_index, -1))

        );

        vector<vector<int>> p2(

            strassen\_multiplication(add(a00, a01, split\_index), b11)

        );

        vector<vector<int>> p3(

            strassen\_multiplication(add(a10, a11, split\_index), b00)

        );

        vector<vector<int>> p4(

            strassen\_multiplication(a11, add(b10, b00, split\_index, -1))

        );

        vector<vector<int>> p5(

            strassen\_multiplication(add(a00, a11, split\_index),add(b00, b11, split\_index))

        );

        vector<vector<int>> p6(

            strassen\_multiplication(add(a01, a11, split\_index, -1),add(b10, b11, split\_index))

        );

        vector<vector<int>> p7(

            strassen\_multiplication(

                add(a00, a10, split\_index, -1),

                add(b00, b01, split\_index)

            )

        );

        // calculating the result

        vector<vector<int> > result\_00(

            add(add(add(p5, p4, split\_index), p6, split\_index), p2, split\_index, -1)

        );

        vector<vector<int> > result\_01(

            add(p1, p2, split\_index)

        );

        vector<vector<int> > result\_10(

            add(p3, p4, split\_index)

        );

        vector<vector<int> > result\_11(

            add(add(add(p5, p1, split\_index), p3, split\_index, -1), p7, split\_index, -1)

        );

        // calulating and storing the result

        // inside matrix

        for (auto i = 0; i < split\_index; i++){

            for (auto j = 0; j < split\_index; j++) {

                result[i][j] = result\_00[i][j];

                result[i][j + split\_index] = result\_01[i][j];

                result[split\_index + i][j] = result\_10[i][j];

                result[i + split\_index][j + split\_index] = result\_11[i][j];

            }

        }

  //------------------------freeing memory------------------------------------------------------//

  //--------------------------Optional done for memory safety----------------------------------//

        a00.clear();

        a01.clear();

        a10.clear();

        a11.clear();

        b00.clear();

        b01.clear();

        b10.clear();

        b11.clear();

        p1.clear();

        p2.clear();

        p3.clear();

        p4.clear();

        p5.clear();

        p6.clear();

        p7.clear();

        result\_00.clear();

        result\_01.clear();

        result\_10.clear();

        result\_11.clear();

    }

    return result;

}

//Driver Code

#include <iostream>

#include <vector>

#include "../basicFunction.h"

#include "strassensMatrixMultiplication.h"

using namespace std;

int main() {

    printinfo();

    vector<vector<int>> matrixA =  {{2, 2, 3, 1},{1, 4, 1, 2},{2, 3, 1, 1}, {1, 3, 1, 2}};

    cout << "Matrix A ==> \n";

    printMatrix(matrixA);

    cout << '\n';

    vector<vector<int>> matrixB = {{2, 1, 2, 1},{3, 1, 2, 1},{3, 2, 1, 1}, {1, 4, 3, 2}};

    cout << "Matrix B ==> \n";

    printMatrix(matrixB);

    cout << '\n';

    vector<vector<int>> matrixResult(strassen\_multiplication(matrixA, matrixB));

    cout << "The result of Strassen matrix Multiplication is: \n";

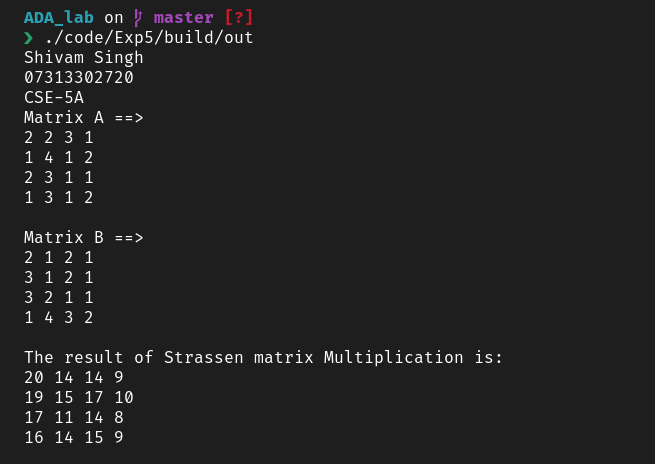
    printMatrix(matrixResult);

    return 0;

}

**Result**

Strassen’s Matrix Multiplication algorithm was implemented using C++

**Output**

**Experiment – 6**

**Aim 🡺 Write a program for Minimum Spanning Tree Algorithms – Prims’ and Kruskal’s Algorithm**

**Software Used 🡺 VSCode, GNU’s C++ compiler**

**Code 🡺**

//Prim’s Algorithm

int minKey(int key[], bool mstSet[])

{

    int min = INT\_MAX, min\_index;

    for (int v = 0; v < V; v++)

        if (mstSet[v] == false && key[v] < min)

            min = key[v], min\_index = v;

    return min\_index;

}

void printMST(int parent[], int graph[V][V])

{

    cout << "Edge \tWeight\n";

    for (int i = 1; i < V; i++)

        cout << parent[i] << " - " << i << " \t"

            << graph[i][parent[i]] << " \n";

}

void primMST(int graph[V][V])

{

    int parent[V];

    int key[V];

    bool mstSet[V];

    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 = minKey(key, mstSet);

        mstSet[u] = true;

        for (int v = 0; v < V; v++)

            if (graph[u][v] && mstSet[v] == false

                && graph[u][v] < key[v])

                parent[v] = u, key[v] = graph[u][v];

    }

    printMST(parent, graph);

}

//Kruskal’s MST Algorithm

class DSU {

    int\* parent;

    int\* rank;

public:

    DSU(int n)

    {

        parent = new int[n];

        rank = new int[n];

        for (int i = 0; i < n; i++) {

            parent[i] = -1;

            rank[i] = 1;

        }

    }

    int find(int i)

    {

        if (parent[i] == -1)

            return i;

        return parent[i] = find(parent[i]);

    }

    void unite(int x, int y)

    {

        int s1 = find(x);

        int s2 = find(y);

        if (s1 != s2) {

            if (rank[s1] < rank[s2]) {

                parent[s1] = s2;

            }

            else if (rank[s1] > rank[s2]) {

                parent[s2] = s1;

            }

            else {

                parent[s2] = s1;

                rank[s1] += 1;

            }

        }

    }

};

class Graph {

    vector<vector<int> > edgelist;

    int v;

public:

    Graph(int v) { this->v = v; }

    void addEdge(int x, int y, int w)

    {

        edgelist.push\_back({ w, x, y });

    }

    void kruskals\_mst()

    {

        sort(edgelist.begin(), edgelist.end());

        DSU s(V);

        int ans = 0;

        cout << "Following are the edges in the "

                "constructed MST"

            << endl;

        for (auto edge : edgelist) {

            int w = edge[0];

            int x = edge[1];

            int y = edge[2];

            if (s.find(x) != s.find(y)) {

                s.unite(x, y);

                ans += w;

                cout << x << " -- " << y << " == " << w

                    << endl;

            }

        }

        cout << "Minimum Cost Spanning Tree: " << ans;

    }

};

//Main Driver Code

#include <bits/stdc++.h>

#include "../basicFunction.h"

#include "primsAlgorithm.h"

#include "kruskalAlgorithm.h"

using namespace std;

int main() {

    printinfo();

    //---------------------------------Prim's Algorithm-------------------------------//

    cout << "\nPrim's MST Algorithms\n";

    /\*  Prim's MST for the following graph

        2 3

    (0)--(1)--(2)

    | / \ |

    6| 8/ \5 |7

    | / \ |

    (3)-------(4)

            9    \*/

    int graph[V][V] = { { 0, 2, 0, 6, 0 },

                        { 2, 0, 3, 8, 5 },

                        { 0, 3, 0, 0, 7 },

                        { 6, 8, 0, 0, 9 },

                        { 0, 5, 7, 9, 0 } };

    // Print the solution

    primMST(graph);

    //------------------------------Kruskal's Algorithm---------------------------------//

    cout << "\nKruskal's MST Algorithms\n";

    /\* Let us create following weighted graph

                10

            0--------1

            | \  |

            6| 5\ |15

            |    \ |

            2--------3

                4    \*/

    Graph g(4);

    g.addEdge(0, 1, 10);

    g.addEdge(1, 3, 15);

    g.addEdge(2, 3, 4);

    g.addEdge(2, 0, 6);

    g.addEdge(0, 3, 5);

    // Function call

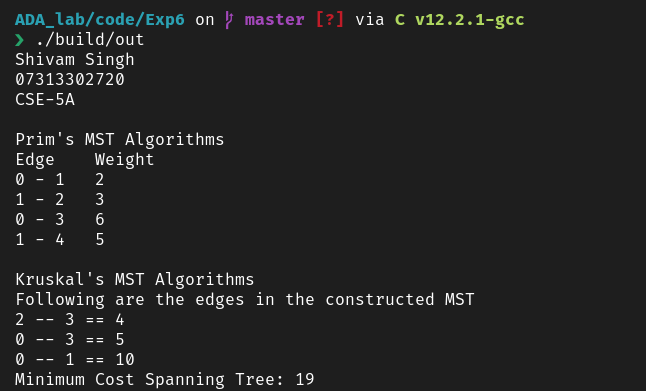
    g.kruskals\_mst();

}

**Result**

Program for Minimum Spanning Tree Algorithms – Prim’s and Kruskal’s Algorithm was implemented using C++

**//Output**



**Experiment – 7**

**Aim 🡺 Write a program for Dijkstra’s Algorithms and Floyd Warshall Algorithm**

**Software Used 🡺 VSCode, GNU’s C++ compiler**

**Code 🡺**

//Dijkstra’s Algorithms

# define INF 0x3f3f3f3f

class Graph

{

    int V;

    list< pair<int, int> > \*adj;

public:

    Graph(int V); // Constructor

    void addEdge(int u, int v, int w);

    void shortestPath(int s);

};

Graph::Graph(int V)

{

    this->V = V;

    adj = new list< pair<int, int> >[V];

}

void Graph::addEdge(int u, int v, int w)

{

    adj[u].push\_back(make\_pair(v, w));

    adj[v].push\_back(make\_pair(u, w));

}

void Graph::shortestPath(int src)

{

    set< pair<int, int> > setds;

    vector<int> dist(V, INF);

    setds.insert(make\_pair(0, src));

    dist[src] = 0;

    while (!setds.empty())

    {

        pair<int, int> tmp = \*(setds.begin());

        setds.erase(setds.begin());

        int u = tmp.second;

        list< pair<int, int> >::iterator i;

        for (i = adj[u].begin(); i != adj[u].end(); ++i)

        {

            int v = (\*i).first;

            int weight = (\*i).second;

            if (dist[v] > dist[u] + weight)

            {

                if (dist[v] != INF)

                    setds.erase(setds.find(make\_pair(dist[v], v)));

                dist[v] = dist[u] + weight;

                setds.insert(make\_pair(dist[v], v));

            }

        }

    }

    printf("Vertex Distance from Source\n");

    for (int i = 0; i < V; ++i)

        printf("%d \t\t %d\n", i, dist[i]);

}

//Floyd-Warshall Algorithms

# define INF 0x3f3f3f3f

class Graph

{

    int V;

    list< pair<int, int> > \*adj;

public:

    Graph(int V); // Constructor

    void addEdge(int u, int v, int w);

    void shortestPath(int s);

};

Graph::Graph(int V)

{

    this->V = V;

    adj = new list< pair<int, int> >[V];

}

void Graph::addEdge(int u, int v, int w)

{

    adj[u].push\_back(make\_pair(v, w));

    adj[v].push\_back(make\_pair(u, w));

}

void Graph::shortestPath(int src)

{

    set< pair<int, int> > setds;

    vector<int> dist(V, INF);

    setds.insert(make\_pair(0, src));

    dist[src] = 0;

    while (!setds.empty())

    {

        pair<int, int> tmp = \*(setds.begin());

        setds.erase(setds.begin());

        int u = tmp.second;

        list< pair<int, int> >::iterator i;

        for (i = adj[u].begin(); i != adj[u].end(); ++i)

        {

            int v = (\*i).first;

            int weight = (\*i).second;

            if (dist[v] > dist[u] + weight)

            {

                if (dist[v] != INF)

                    setds.erase(setds.find(make\_pair(dist[v], v)));

                dist[v] = dist[u] + weight;

                setds.insert(make\_pair(dist[v], v));

            }

        }

    }

    printf("Vertex Distance from Source\n");

    for (int i = 0; i < V; ++i)

        printf("%d \t\t %d\n", i, dist[i]);

}

//Main Driver Code

#include <bits/stdc++.h>

#include "../basicFunction.h"

#include "djikstra.h"

#include "floydWarshal.h"

using namespace std;

int main() {

    printinfo();

    //--------------------------djikstra algorithm-------------------------------------------------//

    cout << "\nDjikstra Algorithm\n";

    int v = 9;

    Graph g(v);

    // making graph

    g.addEdge(0, 1, 4);

    g.addEdge(0, 7, 8);

    g.addEdge(1, 2, 8);

    g.addEdge(1, 7, 11);

    g.addEdge(2, 3, 7);

    g.addEdge(2, 8, 2);

    g.addEdge(2, 5, 4);

    g.addEdge(3, 4, 9);

    g.addEdge(3, 5, 14);

    g.addEdge(4, 5, 10);

    g.addEdge(5, 6, 2);

    g.addEdge(6, 7, 1);

    g.addEdge(6, 8, 6);

    g.addEdge(7, 8, 7);

    g.shortestPath(0);

    //-----------------------------FloydWarshal Algorithm------------------------------//

    cout << "\n FloydWarshal Algorithm\n";

    /\* Let us create the following weighted graph

            10

    (0)------->(3)

        |    /|\

    5 |  |

        |    | 1

    \|/  |

    (1)------->(2)

            3    \*/

    int graph[Vert][Vert] = { { 0, 5, INF, 10 },

                        { INF, 0, 3, INF },

                        { INF, INF, 0, 1 },

                        { INF, INF, INF, 0 } };

    // Function call

    floydWarshall(graph);

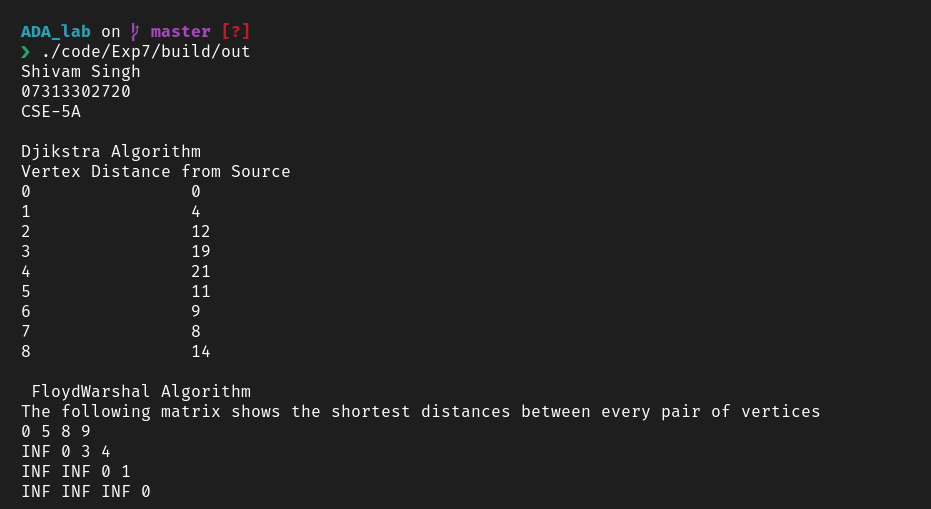
    return 0;

}

**Result**

Program for Minimum Spanning Tree Algorithms – Prim’s and Kruskal’s Algorithms was implemented using C++

**//Output**



**Experiment – 8**

**Aim 🡺 Write a program for 0-1 Knapsack problem**

**Software Used 🡺 VSCode, GNU’s C++ compiler**

**Code 🡺**

//0-1 Knapsack Problem

int max(int a, int b) { return (a > b) ? a : b; }

int knapSack(int W, int wt[], int val[], int n)

{

    if (n == 0 || W == 0)

        return 0;

    if (wt[n - 1] > W)

        return knapSack(W, wt, val, n - 1);

    else

        return max(

            val[n - 1]

                + knapSack(W - wt[n - 1], wt, val, n - 1),

            knapSack(W, wt, val, n - 1));

}

//Main Driver Code

#include <bits/stdc++.h>

#include "../basicFunction.h"

#include "0\_1knapsackProblem.h"

using namespace std;

int main() {

    printinfo();

    cout << "\nThe maximum value by 0/1knapsack method\n";

    int val[] = { 60, 100, 120 };

    int wt[] = { 10, 20, 30 };

    int W = 50;

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

    cout << knapSack(W, wt, val, n);

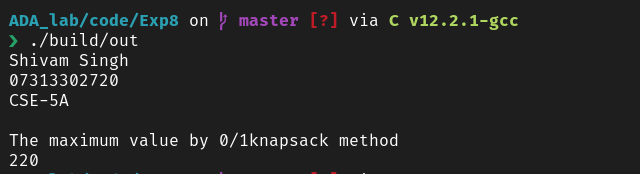
    return 0;

}

**Result**

Program for 0-1 knapsack problem was implemented using C++

**//Output**



**Experiment – 9**

**Aim 🡺 Write a program to implement Longest common subsequence algorithm (LCS)**

**Software Used 🡺 VSCode, GNU’s C++ compiler**

**Code 🡺**

//LCS code

int lcs( char \*X, char \*Y, int m, int n )

{

    if (m == 0 || n == 0)

        return 0;

    if (X[m-1] == Y[n-1])

        return 1 + lcs(X, Y, m-1, n-1);

    else

        return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));

}

//Main Driver Code

#include <bits/stdc++.h>

#include "../basicFunction.h"

#include "LCS.h"

using namespace std;

int main() {

    printinfo();

    char X[] = "AGGTAB";

    char Y[] = "GXTXAYB";

    cout << "String X == " << X << '\n';

    cout << "String Y == " << Y << '\n';

    int m = strlen(X);

    int n = strlen(Y);

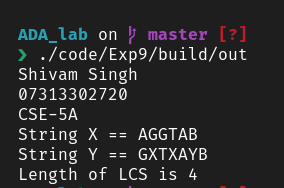
    cout<<"Length of LCS is "<< lcs( X, Y, m, n ) ;

    return 0;

}

**Result**

The program for LCS was implemented using C++

**//Output**

**Experiment – 10**

**Aim 🡺 Write a program to implement Knuth-Morris-Pratt Algorithm**

**Software Used 🡺 VSCode, GNU’s C++ compiler**

**Code 🡺**

//KMP Algorithm

void computeLPSArray(char\* pat, int M, int\* lps);

void KMPSearch(char\* pat, char\* txt)

{

    int M = strlen(pat);

    int N = strlen(txt);

    int lps[M];

    computeLPSArray(pat, M, lps);

    int i = 0; // index for txt[]

    int j = 0; // index for pat[]

    while ((N - i) >= (M - j)) {

        if (pat[j] == txt[i]) {

            j++;

            i++;

        }

        if (j == M) {

            printf("Found pattern at index %d ", i - j);

            j = lps[j - 1];

        }

        else if (i < N && pat[j] != txt[i]) {

            if (j != 0)

                j = lps[j - 1];

            else

                i = i + 1;

        }

    }

}

void computeLPSArray(char\* pat, int M, int\* lps)

{

    int len = 0;

    lps[0] = 0; // lps[0] is always 0

    int i = 1;

    while (i < M) {

        if (pat[i] == pat[len]) {

            len++;

            lps[i] = len;

            i++;

        }

        else // (pat[i] != pat[len])

        {

            if (len != 0) {

                len = lps[len - 1];

            }

            else // if (len == 0)

            {

                lps[i] = 0;

                i++;

            }

        }

    }

}

//Main Driver Code

#include <bits/stdc++.h>

#include "../basicFunction.h"

#include "KnuthMorrisPrath.h"

using namespace std;

int main() {

    printinfo();

    char txt[] = "ABABDABACDABABCABAB";

    char pat[] = "ABABCABAB";

    cout << "The text is: " << txt << '\n';

    cout << "The pattern is: " << pat << '\n';

    KMPSearch(pat, txt);

    return 0;

}

**Result**

Program for Knuth-Morris-Pratt Algorithm was implemented using C++

**//Output**

