Practical-1

AIM: Implement and analyse algorithms given below.

1.1 Factorial (Iterative and Recursive)

1.2 Fibonacci Series (Iterative and Recursive)

1.3 GCD (Iterative and Recursive)

(Make similar pages for all practical, copy practical aim from the practical list.)

1.1 Factorial (Iterative and Recursive)

Program:

*//This program is Prepared by 22CE103 Bhavya*

#include <bits/stdc++.h>

using namespace std;

int f = 1;

int Counter = 1;

int findfac(int x)

{

    Counter += 1;

    if (x == 0 || x == 1)

    {

        return 1;

    }

    return x \* findfac(x - 1);

}

void iterative(int n)

{

    int fac = 1;

    int c = 0;

    c += 3;

    for (int i = 1; i <= n; i++)

    {

        fac = fac \* i;

        c += 3;

    }

    cout << "Factorial:" << fac << endl;

    cout << "Counter is:" << c << endl;

}

int main()

{

    int n;

    cin >> n;

    iterative(n);

    int fac = findfac(n);

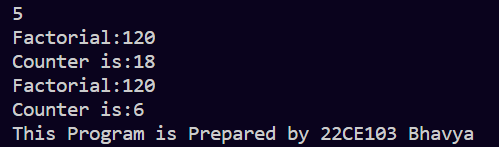
    cout << "Factorial:" << fac << endl;

    cout << "Counter is:" << Counter << endl;

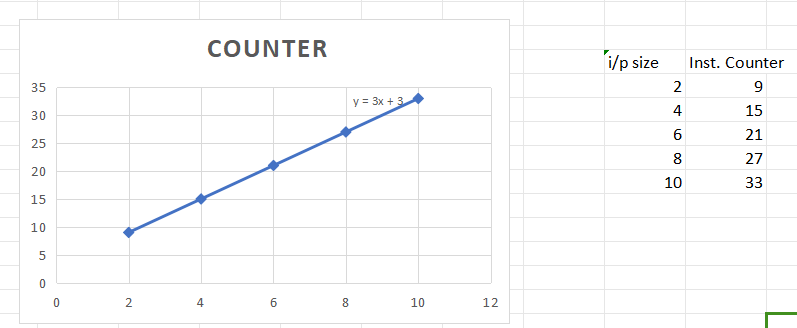
    cout<<"This Program is Prepared by 22CE103 Bhavya";

}

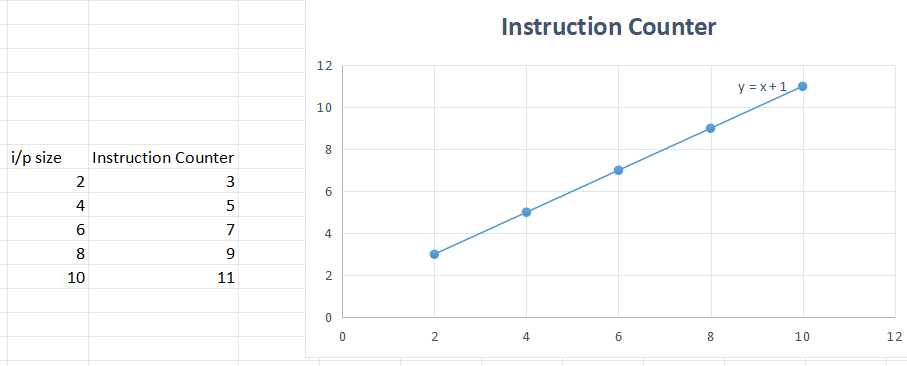
Output:



Iterative:



Recursive:



1.2 Fibonacci Series(Iterative and Recursive)

Program:

#include <bits/stdc++.h>

using namespace std;

int counter = 0;

int counteri=0;

int fibo(int n)

{

    counteri+=1;

    int a = 0, b = 1, c, i;

    if (n == 0)

        return a;

    for (i = 2; i <= n; i++)

    {

        c = a + b;

        a = b;

        b = c;

    }

    return b;

}

int fibbonacci(int n)

{

    counter += 1;

    if (n == 0)

    {

        return 0;

    }

    else if (n == 1)

    {

        return 1;

    }

    else

    {

        return (fibbonacci(n - 1) + fibbonacci(n - 2));

    }

}

int main()

{

    int n;

    cin>>n;

    cout<<"Recursive:"<<fibbonacci(n)<<endl;

    cout<<counter<<endl;

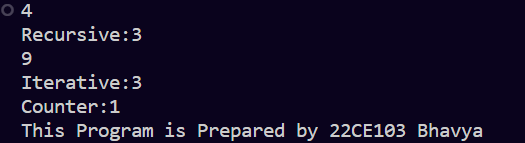
    cout<<"Iterative:"<<fibo(n)<<endl;

    cout<<"Counter:"<<counteri<<endl;

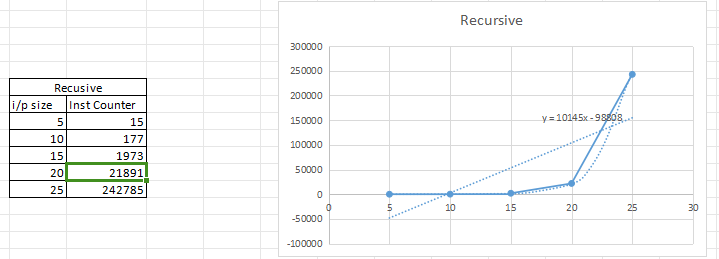
    cout<<"This Program is Prepared by 22CE103 Bhavya";

}

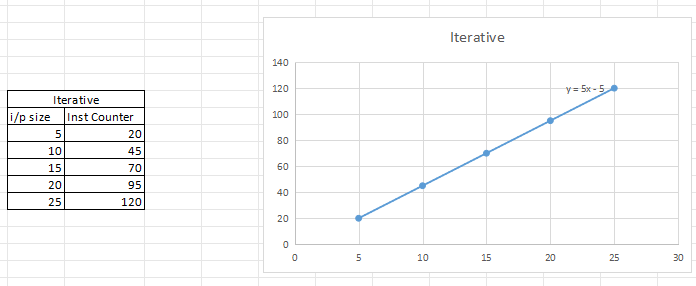
Output:



Iterative:



Recursive:



1.3 GCD (Iterative and Recursive)

Program:

#include<bits/stdc++.h>

using namespace std;

*//iterative GCD*

int GCD(int a, int b){

     int Counter = 0, c;

     Counter = Counter + 2;

     for(int i = a; i >= 1; i--){

          Counter = Counter + 3;

          if(a%i == 0 && b%i == 0){

               c = i;

               Counter = Counter + 1;

               break;

          }

     }

     cout << "Iterative Counter : " << Counter << endl;

     return c;

}

*//recursive GCD (Euclid's Algorithm)*

int Counter = 0;

int recGCD(int a, int b){

     Counter = Counter + 1;

     if(b == 0){

          return a;

     }

     return recGCD(b, a%b);

}

int main(){

     int a, b;

     cin >> a >> b;

     GCD(a,b);

     int result = recGCD(a, b); *//recursive*

     cout<<"Recursive :"<<endl;

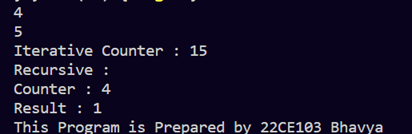
     cout << "Counter : " << Counter << endl;

     cout << "Result : " << result << endl;

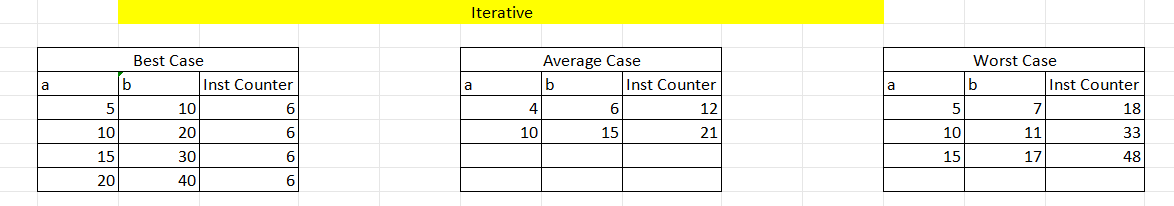
     cout<<"This Program is Prepared by 22CE103 Bhavya";

}

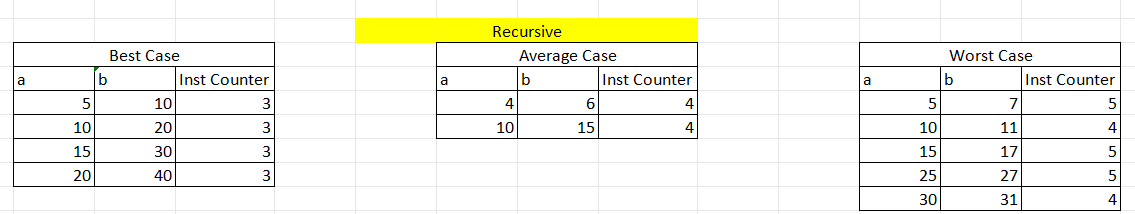
Output:



Iterative:



Recursive:



Practical-2

AIM: Implement and analyse algorithms given below.

2.1 Binary search

2.2 Insertion Sort

2.3 Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution,and you may not use the same element twice. You can return the answer in any order.

2.1 Binary search

Program:

#include <bits/stdc++.h>

using namespace std;

int counter1 = 0;

int counter2 = 0;

int binary\_searchR(int A[], int N, int K, int L, int H)

{

    counter1 += 1;

    while (L <= H)

    {

        int mid = (L + H) / 2;

        if (A[mid] == K)

        {

            cout << "Your Posi Is:" << mid << endl;

            break;

        }

        else if (A[mid] < K)

        {

            cout << A[mid] << " " << endl;

            binary\_searchR(A, N, K, mid + 1, H);

            break;

        }

        else

        {

            binary\_searchR(A, N, K, L, mid - 1);

            break;

        }

    }

}

int binarySearchI(int arr[], int l, int r, int x)

{

    counter2 += 3;

    while (l <= r)

    {

        int m = l + (r - l) / 2;

        counter2 += 2;

        if (arr[m] == x)

            cout << "Iterative Counter:" << counter2 << endl;

        return m;

        if (arr[m] < x)

            l = m + 1;

        else

            r = m - 1;

        counter2 += 2;

    }

    return -1;

}

int main()

{

    int c, l = 0, h = 5, key;

    int arr[5];

    cout << "Enter Your Key Element: ";

    cin >> key;

    cout << "Enter Array Element:" << endl;

    for (int i = 0; i < 5; i++)

    {

        cin >> arr[i];

    }

    binary\_searchR(arr, 5, key, l, h);

    int result = binarySearchI(arr, 0, 4, key);

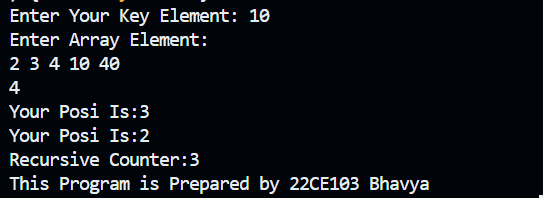
    cout << "Your Posi Is:" << result << endl;

    cout << "Recursive Counter:" << counter1 << endl;

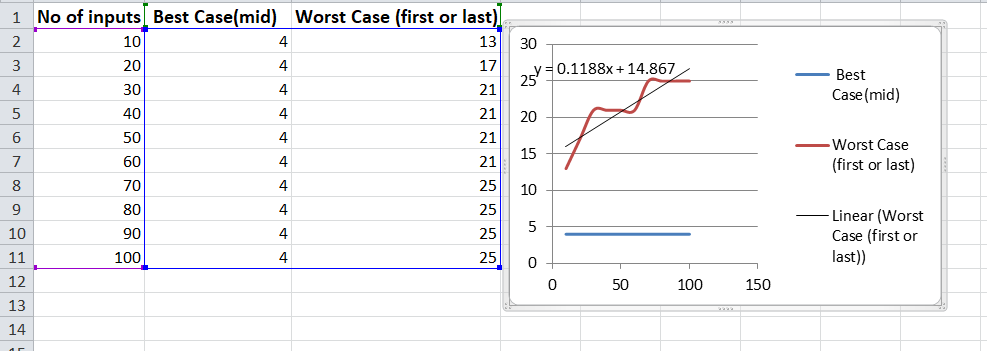
    cout<<"This Program is Prepared by 22CE103 Bhavya"<<endl;

}

Output:



Graph:

****

2.1 Insertion Sort

Program:

#include<bits/stdc++.h>

using namespace std;

int Counter=0;

int main()

{

    int n,key;

    int arr[20];

    cout<<"Enter Size of array: ";

    cin>>n;

    cout<<"Enter Array:"<<endl;

    for(int i=1;i<=n;i++)

    {

        cin>>arr[i];

    }

    //+3

    Counter+=2;

    for(int i=2;i<=n;i++)

    {

        key=arr[i];

        int j=i-1;

        //+2

        Counter+=2;

        while(j>0 && arr[j]>key)

        {

            //+2

            Counter+=2;

            arr[j+1]=arr[j];

            j=j-1;

            //+2

            Counter+=2;

        }

        arr[j+1]=key;

        //+3

        Counter+=3;

    }

    cout<<endl;

    for(int i=1;i<=n;i++)

    {

        cout<<arr[i]<<" ";

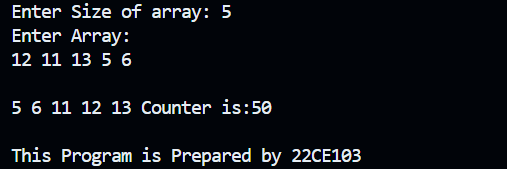
    }

    cout<<"Counter is:"<<Counter<<endl;

    cout<<"\nThis Program is Prepared by 22CE103"<<endl;

}

Output:



2.3 Two Sum

Program:

class Solution {

public:

    vector<int> twoSum(vector<int>& nums, int target) {

        map<int,int> m;

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

        {

            int temp=target-nums[i];

            if(m.find(temp)==m.end())

            {

                m[nums[i]]=i;

            }

            else

            {

                return {m[temp],i};

            }

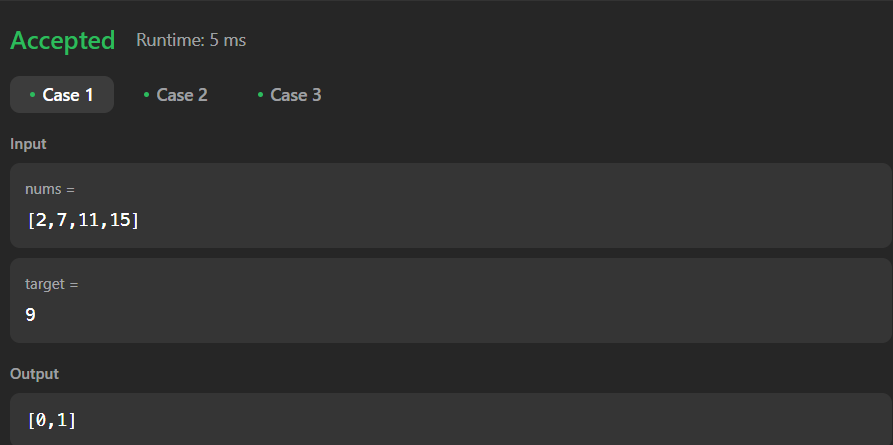
        }

        return {-1, -1};

    }

};

Output:



Practical-4

AIM: Greedy Approach.

4.1 A Burglar has just broken into the Fort! He sees himself in a room with n piles of gold dust. Because each pile has a different purity, each pile also has a different value (v[i]) and a different weight (w[i]). A Burglar has a bag that can only hold W kilograms. Calculate which piles Burglar should completely put into his bag and which he should put only fraction into his bag. Design and implement an algorithm to get maximum piles of gold using given bag with W capacity, Burglar is also allowed to take fractional of pile.

4.2 Implement the program to find the shortest path from one source to all other destinations in any city graph.

4.3 Find Minimum Cost spanning tree of an undirected graph using Kruskal’s algorithm.

4.1 Knapsack Fractional Problem

Program:

#include<bits/stdc++.h>

using namespace std;

int main() {

    vector<int> values;

    vector<int> weight;

    vector<int> item;

    vector<double> frac;

    cout << "Enter Size of Input: ";

    int size, temp;

    cin >> size;

    cout << "Enter Values: ";

    for (int i = 0; i < size; i++) {

        cin >> temp;

        values.push\_back(temp);

    }

    cout << "Enter Weight: ";

    for (int i = 0; i < size; i++) {

        cin >> temp;

        weight.push\_back(temp);

    }

    cout << "Enter Item: ";

    for (int i = 0; i < size; i++) {

        cin >> temp;

        item.push\_back(temp);

    }

    for (int i = 0; i < size; i++) {

        double tempo = (double)values[i] / weight[i];

        frac.push\_back(tempo);

    }

    for (int i = 0; i < size; i++) {

        for (int j = i + 1; j < size; j++) {  // Fix: Change i<size to j<size

            if (frac[j] > frac[i]) {

                swap(values[j], values[i]);

                swap(weight[j], weight[i]);

                swap(item[j], item[i]);

                swap(frac[j], frac[i]);

            }

        }

    }

    int iter=0,profit=0;

    int W=50;

    int X[size];

    while(W!=0)

    {

        if(weight[iter]<=W)

        {

            X[item[iter]-1]=1;

            profit+=values[iter];

            W=W-weight[iter];

        }

        else

        {

            float fr=(double)(W)/weight[iter];

            profit+=fr\*values[iter];

            W=0;

            X[item[iter]-1]=fr;

        }

        iter+=1;

    }

    cout<<"Profit:"<<profit<<endl;

    cout<<"The Set is:"<<endl;

    cout<<X[0]<<" "<<X[1]<<" "<<X[2]<<endl;

    cout << "Sorted Values:\n";

    for (int i = 0; i < size; i++) {

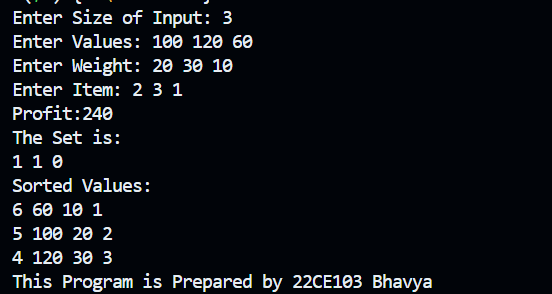
        cout << frac[i] << " " << values[i] << " " << weight[i] << " " << item[i] << endl;

    }

    cout<<"This Program is Prepared by 22CE103 Bhavya"<<endl;

}

Output:



4.2 Dijkstra’s Algorithm

Program:

#include <bits/stdc++.h>

using namespace std;

void Dijkstra(vector<vector<int>> &graph, vector<int> &distance, vector<int> &vis, int src)

{

    int n = graph.size();

    distance[src] = 0;

    for (int i = 0; i < n - 1; ++i)

    {

        int u = -1;

        for (int j = 0; j < n; ++j)

        {

            if (!vis[j] && (u == -1 || distance[j] < distance[u]))

            {

                u = j;

            }

        }

        vis[u] = 1;

        for (int v = 0; v < n; ++v)

        {

            if (graph[u][v] && !vis[v])

            {

                distance[v] = min(distance[v], distance[u] + graph[u][v]);

            }

        }

    }

}

int main()

{

    int size;

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

    cin >> size;

    vector<int> distance(size, INT\_MAX);

    vector<int> vis(size, 0);

    int src;

    cout << "Enter Your Source node: ";

    cin >> src;

    vector<vector<int>> graph;

    for (int i = 0; i < size; i++)

    {

        vector<int> temp;

        for (int j = 0; j < size; j++)

        {

            int value;

            cin >> value;

            temp.push\_back(value);

        }

        graph.push\_back(temp);

    }

    Dijkstra(graph, distance, vis, src);

    cout << "Shortest distances from node " << src << " to all other nodes:" << endl;

    for (int i = 0; i < size; ++i)

    {

        cout << "Node " << i << ": " << distance[i] << endl;

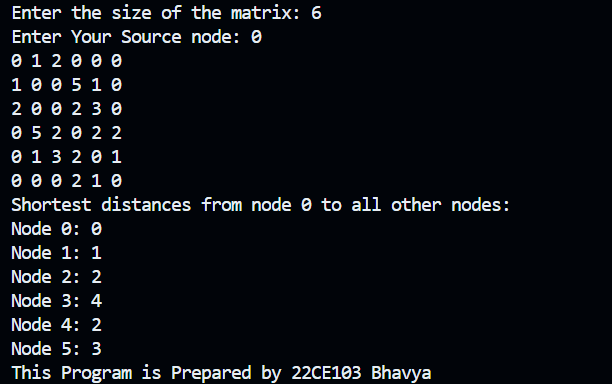
    }

    cout << "This Program is Prepared by 22CE103 Bhavya" << endl;

    return 0;

}

Output:



4.3 Kruskal’s Algorithm

Program:

#include <bits/stdc++.h>

using namespace std;

struct graph

{

    int first, second;

    int weight;

};

int find(int element, vector<int> &parent)

{

    if (parent[element] == element)

    {

        return element;

    }

    else

    {

        return find(parent[element], parent);

    }

}

void Union(int x, int y, vector<int> &parent)

{

    int x\_set\_parent = find(x, parent);

    int y\_set\_parent = find(y, parent);

    parent[y\_set\_parent] = x\_set\_parent;

}

void kruskal(vector<int> &parent, graph edges[], int size) // vector<graph> edges

{

    int cost = 0;

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

    {

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

        {

            if (edges[j].weight > edges[j + 1].weight)

            {

                swap(edges[j], edges[j + 1]);

            }

        }

    }

    for (int i = 0; i < size; i++)

    {

        int a = find(edges[i].first, parent);

        int b = find(edges[i].second, parent);

        if (a != b)

        {

            cout << edges[i].first << " --- " << edges[i].second << endl;

            cost += edges[i].weight;

            Union(a, b, parent);

        }

    }

    cout << "Total cost= " << cost << endl;

}

int main()

{

    int size;

    cout << "Enter Number of Edges:";

    cin >> size;

    graph edges[size]; // vector<graph> edges(size);

    vector<int> parent(size + 1);

    cout << "Enter details for each edge (first second weight):" << endl;

    for (int i = 0; i < size; i++)

    {

        cin >> edges[i].first >> edges[i].second >> edges[i].weight;

    }

    for (int i = 1; i <= size; i++)

    {

        parent[i] = i;

    }

    cout << "Edges:" << endl;

    for (int i = 0; i < size; i++)

    {

        cout << "Edge " << i + 1 << ": " << edges[i].first << " -> " << edges[i].second

             << ", Weight: " << edges[i].weight << endl;

    }

    cout << "Initial parent array:" << endl;

    for (int i = 1; i <= size; i++)

    {

        cout << parent[i] << " ";

    }

    cout << endl;

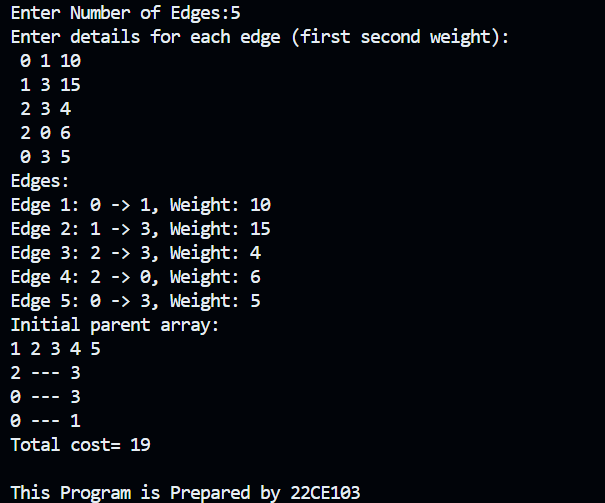
    kruskal(parent, edges, size); // kruskal(parent,edges)

    cout<<"\nThis Program is Prepared by 22CE103"<<endl;

    return 0;

}

Output:



4.4 Best Time to Buy and Sell Stock

Program:

class Solution {

public:

    int maxProfit(vector<int>& prices) {

        int min\_price = prices[0];

        int maxprof = 0;

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

            maxprof = max(maxprof,prices[i]-min\_price);

            min\_price = min(prices[i],min\_price);

        }

        return maxprof;

    }

};

Output:

