**Lab File**

**Analysis and Design of Algorithms**

**(CSE 303)**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

A blue and yellow logo

Description automatically generated with low confidence

Submitted to: Submitted by:

Dr Richa Tiwari Shaina Mehta

Ast. Professor A2305219268

CSE Department, ASET B.tech. C.S.E.

5CSE-4Y

Amity School Of Engineering and technology

Amity University Uttar Pradesh

Noida -201301

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Exp No | Assignment  Category | Code | Name of Experiment | Date of Allotment | Date of Evaluation | Max  Marks | Marks  Obtained | Faculty  Sign |
| 1 | Mandatory  Experiment |  | To implement bubble sort. | 21-07-2021 | 08-09-2021 |  |  |  |
| 2 |  | To implement insertion sort algorithm and plot the graph on its execution time. | 28-07-2021 | 08-09-2021 |  |  |  |
| 3 |  | To implement merge sort. | 4-08-2021 | 08-09-2021 |  |  |  |
| 4 |  | To implement quick sort. | 11-08-2021 | 08-09-2021 |  |  |  |
| 5 |  | To implement Strassen’s matrix multiplication. | 18-08-2021 | 08-09-2021 |  |  |  |
| 6 |  | To analyse the time complexity of Bubble Sort, Merge Sort, Quick Sort and Insertion Sort algorithms using emprical method and plotting the graph for the same. | 25-08-2021 | 08-09-2021 |  |  |  |
| 7 |  | To create MST of a graph using Prim’s Algorithm. | 01-09-2021 | 08-09-2021 |  |  |  |
| 8 |  |  | To create MST of a graph using Kruskal’s Algorithm. | 08-09-2021 | 08-09-2021 |  |  |  |
|  | Viva | Viva |  |  |  |  |  |  |

**Experiment 1**

**Aim:** To implement bubble sort.

**Code:**

1. **Worst Case**

#include <iostream>

using namespace std;

int pass=0, comp=0, swaps=0;

void Bubble\_Sort(int \*arr, int n){

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

pass++;

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

comp++;

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

swaps++;

int temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

cout<<"The array after pass "<<i+1<<" is: ";

for(int k=0;k<n;k++){

cout<<arr[k]<<" ";

}

cout<<endl;

}

}

int main(){

int n;

cout<<"Bubble Sort Algorithm"<<endl;

cout<<"Best Case Time Complexity is: O(n)"<<endl;

cout<<"Worst Case Time Complexity is: O(n^2)"<<endl;

cout<<"Space Complexity is: O(n)"<<endl;

cout<<"Enter the size of an array (Total entries should not be more than 100): ";

cin>>n;

int \*arr = new int [100];

cout<<"Enter the array elements: ";

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

cin>>arr[i];

}

cout<<"The original array is: ";

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

cout<<arr[i]<<" ";

}

cout<<endl;

Bubble\_Sort(arr,n);

cout<<"The sorted array in ascending order is: ";

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

cout<<arr[i]<<" ";

}

cout<<endl;

cout<<"No of passes is: "<<pass<<endl;

cout<<"No of comparisons is: "<<comp<<endl;

cout<<"No of swaps is: "<<swaps<<endl;

delete arr;

}

1. **Best Case**

#include <iostream>

using namespace std;

int pass=0, comp=0, swaps=0;

void Bubble\_Sort(int \*arr, int n){

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

pass++;

cout<<"The array after pass "<<i+1<<" is: ";

for(int k=0;k<n;k++){

cout<<arr[k]<<" ";

}

cout<<endl;

}

}

int main(){

int n;

cout<<"Bubble Sort Algorithm"<<endl;

cout<<"Best Case Time Complexity is: O(n)"<<endl;

cout<<"Worst Case Time Complexity is: O(n^2)"<<endl;

cout<<"Space Complexity is: O(n)"<<endl;

cout<<"Enter the size of an array (Total entries should not be more than 100): ";

cin>>n;

int \*arr = new int [100];

cout<<"Enter the array elements: ";

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

cin>>arr[i];

}

cout<<"The original array is: ";

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

cout<<arr[i]<<" ";

}

cout<<endl;

Bubble\_Sort(arr,n);

cout<<"The sorted array in ascending order is: ";

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

cout<<arr[i]<<" ";

}

cout<<endl;

cout<<"No of passes is: "<<pass<<endl;

cout<<"No of comparisons is: "<<comp<<endl;

cout<<"No of swaps is: "<<swaps<<endl;

delete arr;

}

**Output:**

1. **Best Case**

Text

Description automatically generated

1. **Worst Case**

Text

Description automatically generated

**Conclusion:** The bubble sort sorting algorithm has been implemented successfully.

**Experiment 2**

**Aim:** To implement insertion sort algorithm and plot the graph on its execution time.

**Code:**

#include <iostream>

#include <ctime>

using namespace std;

int pass=0,comp=0,swaps=0;

void Insertion\_Sort(int \*a, int n){

for(int i=1;i<n;i++){

pass++;

int temp = a[i];

int j = i-1;

while(j>=0){

comp++;

if(temp<a[j]){

a[j+1]=a[j];

swaps++;

}

else{

break;

}

j--;

}

a[j+1]=temp;

}

}

int main(){

int n;

clock\_t s,f;

cout<<"Insertion Sort Algorithm"<<endl;

cout<<"Best Case Time Complexity is: O(n)"<<endl;

cout<<"Worst Case Time Complexity is: O(n^2)"<<endl;

cout<<"Space Complexity is: O(n)"<<endl;

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

cin>>n;

int \*arr = new int [100];

cout<<"Enter the array elements: ";

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

cin>>arr[i];//=rand()%100;

}

cout<<"The original array is: ";

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

cout<<arr[i]<<" ";

}

cout<<endl;

s=clock();

//cout<<s;

Insertion\_Sort(arr,n);

f=clock();

//cout<<" "<<f<<endl;

cout<<"The sorted array in ascending order is: ";

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

cout<<arr[i]<<" ";

}

cout<<endl;

cout<<"No of passes is: "<<pass<<endl;

cout<<"No of comparisons is: "<<comp<<endl;

cout<<"No of swaps is: "<<swaps<<endl;

//cout<<"The time required for insertion sort is: "<< ((double)(f-s))/CLOCKS\_PER\_SEC <<" seconds."<<endl;

delete arr;

}

**Output:**

1. **Best Case:**

**Text

Description automatically generated**

1. **Worst Case:**

**Text

Description automatically generated**

**Analysis:**

|  |  |
| --- | --- |
| Entries | Time Taken (seconds) |
| 10 | 0 |
| 50 | 0 |
| 100 | 0 |
| 500 | 0 |
| 1000 | 0 |
| 5000 | 0.015 |
| 10000 | 0.1 |
| 50000 | 2.21 |
| 100000 | 8.134 |
| 500000 | 233.036 |

**Chart, line chart

Description automatically generated**

**Conclusion:** The insertion sort sorting algorithm has been implemented and analysed successfully.

**Experiment 3**

**Aim:** To implement merge sort.

**Code:**

#include <iostream>

using namespace std;

void Merge(int \*arr, int si, int ei, int mid){

int a[100],b[100],c[100];

int m=0,n=0;

for(int i=si;i<=mid;i++){

a[m]=arr[i];

m++;

}

for(int i=mid+1;i<=ei;i++){

b[n]=arr[i];

n++;

}

int j=0,k=0,l=0;

while(j<m && k<n){

if(a[j]<=b[k]){

c[l]=a[j];

l++;

j++;

}

else{

c[l]=b[k];

l++;

k++;

}

}

if(j==m){

while(k<=n){

c[l]=b[k];

l++;

k++;

}

}

else{

while(j<=m){

c[l]=a[j];

l++;

j++;

}

}

int p=0;

for(int i=si;i<=ei;i++){

arr[i]=c[p];

p++;

}

}

void Split(int \*arr,int si,int ei){

if(si>=ei){

return;

}

int mid=(si+ei)/2;

Split(arr,si,mid);

Split(arr,mid+1,ei);

Merge(arr,si,ei,mid);

}

void Merge\_Sort(int \*arr, int n){

int si=0,ei=n-1;

Split(arr,si,ei);

}

int main(){

int n;

cout<<"Merge Sort Algorithm"<<endl;

cout<<"Best Case Time Complexity is: O(n\*logn)"<<endl;

cout<<"Worst Case Time Complexity is: O(n\*logn)"<<endl;

cout<<"Space Complexity is: O(n)"<<endl;

cout<<"Enter the size of an array (Total entries should not be more than 100): ";

cin>>n;

int \*arr = new int [100];

cout<<"Enter the array elements: ";

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

cin>>arr[i];

}

cout<<"The original array is: ";

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

cout<<arr[i]<<" ";

}

cout<<endl;

Merge\_Sort(arr,n);

cout<<"The sorted array in ascending order is: ";

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

cout<<arr[i]<<" ";

}

cout<<endl;

delete arr;

}

**Output:**

1. **Best Case**

Text

Description automatically generated

1. **Worst Case**

Text

Description automatically generated

**Conclusion:** The merge sort sorting algorithm has been implemented successfully.

**Experiment 4**

**Aim:** To implement quick sort.

**Code:**

#include <iostream>

using namespace std;

int Pivot(int \*arr, int si, int ei){

int x = arr[si];

int c = 0;

for(int i=si+1; i<=ei; i++){

if(x>=arr[i]){

c++;

}

}

int p = si+c;

int temp = arr[si];

arr[si] = arr[p];

arr[p] = temp;

int i = si, j = ei;

while(i<p && j>p){

if(arr[i] <= x){

i++;

}

else if(arr[j] > x){

j--;

}

else{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

i++;

j--;

}

}

return p;

}

void Quick(int \*arr, int si, int ei){

if(si >= ei){

return;

}

int c = Pivot(arr,si,ei);

Quick(arr,si,c-1);

Quick(arr,c+1,ei);

}

void Quick\_Sort(int \* arr, int n){

int si = 0, ei = n-1;

Quick(arr,si,ei);

}

int main(){

int n;

cout<<"Quick Sort Algorithm"<<endl;

cout<<"Best Case Time Complexity is: O(n\*logn)"<<endl;

cout<<"Worst Case Time Complexity is: O(n^2)"<<endl;

cout<<"Space Complexity is: O(n)"<<endl;

cout<<"Enter the size of an array (Total entries should not be more than 100): ";

cin>>n;

int \*arr = new int [100];

cout<<"Enter the array elements: ";

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

cin>>arr[i];

}

cout<<"The original array is: ";

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

cout<<arr[i]<<" ";

}

cout<<endl;

Quick\_Sort(arr,n);

cout<<"The sorted array in ascending order is: ";

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

cout<<arr[i]<<" ";

}

cout<<endl;

delete arr;

}

**Output:**

1. **Best Case:**

A picture containing text, screenshot, computer, monitor

Description automatically generated

1. **Worst Case:**

A picture containing text, screenshot, computer, monitor

Description automatically generated

**Conclusion:** The quick sort sorting algorithm has been implemented successfully.

**Experiment 5**

**Aim:** To implement Strassen’s matrix multiplication.

**Code:**

import random as rn

def add\_mat(A,B,N):

S = [[0 for i in range(N)] for i in range(N)]

for i in range(N):

for j in range(N):

S[i][j] = A[i][j] + B[i][j]

return S

def sub\_mat(A,B,N):

S = [[0 for i in range(N)] for i in range(N)]

for i in range(N):

for j in range(N):

S[i][j] = A[i][j] - B[i][j]

return S

def s\_multiplication(A,B,N):

C = [[0 for i in range(N)] for i in range(N)]

if(N == 1):

return A[0][0]\*B[0][0]

else:

mid = N//2

A11 = [[A[i][j] for j in range(0,mid)] for i in range(0,mid)]

A12 = [[A[i][j] for j in range(mid,N)] for i in range(0,mid)]

A21 = [[A[i][j] for j in range(0,mid)] for i in range(mid,N)]

A22 = [[A[i][j] for j in range(mid,N)] for i in range(mid,N)]

B11 = [[B[i][j] for j in range(0,mid)] for i in range(0,mid)]

B12 = [[B[i][j] for j in range(mid,N)] for i in range(0,mid)]

B21 = [[B[i][j] for j in range(0,mid)] for i in range(mid,N)]

B22 = [[B[i][j] for j in range(mid,N)] for i in range(mid,N)]

S1 = sub\_mat(B12,B22,mid)

S2 = add\_mat(A11,A12,mid)

S3 = add\_mat(A21,A22,mid)

S4 = sub\_mat(B21,B11,mid)

S5 = add\_mat(A11,A22,mid)

S6 = add\_mat(B11,B22,mid)

S7 = sub\_mat(A12,A22,mid)

S8 = add\_mat(B21,B22,mid)

S9 = sub\_mat(A11,A21,mid)

S10 = add\_mat(B11,B12,mid)

P1 = s\_multiplication(A11,S1,mid)

P2 = s\_multiplication(S2,B22,mid)

P3 = s\_multiplication(S3,B11,mid)

P4 = s\_multiplication(A22,S4,mid)

P5 = s\_multiplication(S5,S6,mid)

P6 = s\_multiplication(S7,S8,mid)

P7 = s\_multiplication(S9,S10,mid)

#print(P1,P2,P3,P4,P5,P6,P7)

if(mid == 1):

C[0][0] = P5 + P4 - P2 + P6

C[0][1] = P1 + P2

C[1][0] = P3 + P4

C[1][1] = P5 + P1 - P3 - P7

#print(C)

return C

C11 = [[0 for j in range(0,mid)] for i in range(0,mid)]

C12 = [[0 for j in range(0,mid)] for i in range(0,mid)]

C21 = [[0 for j in range(0,mid)] for i in range(0,mid)]

C22 = [[0 for j in range(0,mid)] for i in range(0,mid)]

for i in range(0,mid):

for j in range(0,mid):

C11[i][j] = P5[i][j] + P4[i][j] - P2[i][j] + P6[i][j]

C12[i][j] = P1[i][j] + P2[i][j]

C21[i][j] = P3[i][j] + P4[i][j]

C22[i][j] = P5[i][j] + P1[i][j] - P3[i][j] - P7[i][j]

for i in range(0,mid):

for j in range(0,mid):

C[i][j] = C11[i][j]

C[i][j+mid] = C12[i][j]

C[i+mid][j] = C21[i][j]

C[i+mid][j+mid] = C22[i][j]

#print(C)

return C

print('Enter the size of the matrix: ')

N = int(input())

A = [[rn.randint(1,9) for j in range(N)] for i in range(N)]

B = [[rn.randint(1,9) for j in range(N)] for i in range(N)]

print('First Matrix: ')

for i in range(N):

for j in range(N):

print(A[i][j], end = " ")

print(end = '\n')

print('Second Matrix: ')

for i in range(N):

for j in range(N):

print(B[i][j], end = " ")

print(end = '\n')

C = s\_multiplication(A,B,N)

print('Resultant Matrix: ')

for i in range(N):

for j in range(N):

print(C[i][j], end = " ")

print(end = '\n')

**Output:**

**Text

Description automatically generated**

**Conclusion:** Strassen’s Matrix Multiplication algorithm has been implemented successfully.

**Experiment 6**

**Aim:** To analyse the time complexity of Bubble Sort, Merge Sort, Quick Sort and Insertion Sort algorithms using emprical method and plotting the graph for the same.

**Procedure:** This experiment is performed as follows:

1) Implementing all the algorithms using random integers from 1 to 100 and measuring the time taken for all the sorting algorithms implemented in experiment 1, 2, 3 and 4 using clock function in C++ for the entries between the size 10 - 100000 in the.csv file.

2) Plotting the graph using numpy, pandas and matplotlib library in Python.

**Code and Output:**

**#Importing Necessary Libraries**

import numpy as np

**[1]** import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

**#Take the Input from the Sorting Algorithms.csv**

**[2]** algo = pd.read\_csv('C://Users//hp//Desktop//Shaina//Sorting Algorithms.csv')

**[3]** algo

**Out [3]**

Table

Description automatically generated

**[4]** labels = algo['Entries']

**[5]** labels = list(np.array(labels))

**[6]** labels

**Out [6]**



**[7]** B\_Sort = list(np.array(algo['Bubble Sort']))

I\_Sort = list(np.array(algo['Insertion Sort']))

M\_Sort = list(np.array(algo['Merge Sort']))

Q\_Sort = list(np.array(algo['Quick Sort']))

**# Plotting the Graph**

**[8]** plt.plot(labels,B\_Sort,'--b')

plt.plot(labels,I\_Sort,'-og')

plt.plot(labels,M\_Sort,'oy')

plt.plot(labels,Q\_Sort,'--r')

plt.xlabel('Time in Milliseconds')

plt.ylabel('Input Size / Number of Elements in Array')

plt.title('Sorting Algorithms Comparisons')

plt.legend(['Bubble','Insertion','Merge','Quick'])

**Out [8]**

Chart, line chart

Description automatically generated

**Results and Conclusion:** The analysis of the time complexity of Bubble Sort, Merge Sort, Quick Sort and Insertion Sort algorithms using empirical method and graph plotting has been done successfully.

https://github.com/shaina-12/Data-Structures-And-Algorithms-Codes/blob/d47e9a8ecdd3d92bc784afbe5dca1bb5d6980ce4/Sorting%20Algorithms/Sorting%20Algorithms.ipynb

**Experiment 7**

**Aim:** To create MST of a graph using Prim’s Algorithm.

**Code:**

// A C++ program for Prim's Minimum

// Spanning Tree (MST) algorithm. The program is

// for adjacency matrix representation of the graph

#include <bits/stdc++.h>

using namespace std;

// Number of vertices in the graph

#define V 5

// A utility function to find the vertex with

// minimum key value, from the set of vertices

// not yet included in MST

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

{

// Initialize min value

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;

}

// A utility function to print the

// constructed MST stored in parent[]

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";

}

// Function to construct and print MST for

// a graph represented using adjacency

// matrix representation

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

{

// Array to store constructed MST

int parent[V];

// Key values used to pick minimum weight edge in cut

int key[V];

// To represent set of vertices included in MST

bool mstSet[V];

// Initialize all keys as INFINITE

for (int i = 0; i < V; i++)

key[i] = INT\_MAX, mstSet[i] = false;

// Always include first 1st vertex in MST.

// Make key 0 so that this vertex is picked as first vertex.

key[0] = 0;

parent[0] = -1; // First node is always root of MST

// The MST will have V vertices

for (int count = 0; count < V - 1; count++)

{

// Pick the minimum key vertex from the

// set of vertices not yet included in MST

int u = minKey(key, mstSet);

// Add the picked vertex to the MST Set

mstSet[u] = true;

// Update key value and parent index of

// the adjacent vertices of the picked vertex.

// Consider only those vertices which are not

// yet included in MST

for (int v = 0; v < V; v++)

// graph[u][v] is non zero only for adjacent vertices of m

// mstSet[v] is false for vertices not yet included in MST

// Update the key only if graph[u][v] is smaller than key[v]

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

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

}

// print the constructed MST

printMST(parent, graph);

}

// Driver code

int main()

{

/\* Let us create 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);

return 0;

}

**Output:**

**Text

Description automatically generated**

**Results and Conclusion:** MST of a graph using Prim’s Algorithm has been done successfully.

**Experiment 8**

**Aim:** To create MST of a graph using Kruskal’s Algorithm.

**Code:**

// C++ program for Kruskal's algorithm to find Minimum

// Spanning Tree of a given connected, undirected and

// weighted graph

#include<bits/stdc++.h>

using namespace std;

// Creating shortcut for an integer pair

typedef pair<int, int> iPair;

// Structure to represent a graph

struct Graph

{

int V, E;

vector< pair<int, iPair> > edges;

// Constructor

Graph(int V, int E)

{

this->V = V;

this->E = E;

}

// Utility function to add an edge

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

{

edges.push\_back({w, {u, v}});

}

// Function to find MST using Kruskal's

// MST algorithm

int kruskalMST();

};

// To represent Disjoint Sets

struct DisjointSets

{

int \*parent, \*rnk;

int n;

// Constructor.

DisjointSets(int n)

{

// Allocate memory

this->n = n;

parent = new int[n+1];

rnk = new int[n+1];

// Initially, all vertices are in

// different sets and have rank 0.

for (int i = 0; i <= n; i++)

{

rnk[i] = 0;

//every element is parent of itself

parent[i] = i;

}

}

// Find the parent of a node 'u'

// Path Compression

int find(int u)

{

/\* Make the parent of the nodes in the path

from u--> parent[u] point to parent[u] \*/

if (u != parent[u])

parent[u] = find(parent[u]);

return parent[u];

}

// Union by rank

void merge(int x, int y)

{

x = find(x), y = find(y);

/\* Make tree with smaller height

a subtree of the other tree \*/

if (rnk[x] > rnk[y])

parent[y] = x;

else // If rnk[x] <= rnk[y]

parent[x] = y;

if (rnk[x] == rnk[y])

rnk[y]++;

}

};

/\* Functions returns weight of the MST\*/

int Graph::kruskalMST()

{

int mst\_wt = 0; // Initialize result // Sort edges in increasing order on basis of cost

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

// Create disjoint sets

DisjointSets ds(V);

// Iterate through all sorted edges

vector< pair<int, iPair> >::iterator it;

for (it=edges.begin(); it!=edges.end(); it++)

{

int u = it->second.first;

int v = it->second.second;

int set\_u = ds.find(u);

int set\_v = ds.find(v);

// Check if the selected edge is creating

// a cycle or not (Cycle is created if u

// and v belong to same set)

if (set\_u != set\_v)

{

// Current edge will be in the MST

// so print it

cout << u << " - " << v << endl;

// Update MST weight

mst\_wt += it->first;

// Merge two sets

ds.merge(set\_u, set\_v);

}

}return mst\_wt;

}

// Driver program to test above functions

int main()

{

/\* Let us create above shown weighted

and unidrected graph \*/

int V = 9, E = 14;

Graph g(V, E);

// making above shown 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);

cout << "Edges of MST are \n";

int mst\_wt = g.kruskalMST();

cout << "\nWeight of MST is " << mst\_wt;

return 0;

}

**Output:**

**Text

Description automatically generated**

**Results and Conclusion:** MST of a graph using Kruskal’s Algorithm has been done successfully.

For More Codes:

<https://github.com/shaina-12/Data-Structures-And-Algorithms-Codes.git>