Codes

**1. Write a program to implement and analyse time complexity of Insertion sort & Selection Sort**

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

void insertionSort(int arr[], int n)

{

int i, key, j;

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

{

key = arr[i];

j = i - 1;

while (j >= 0 && arr[j] > key)

{

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

j = j - 1;

}

arr[j + 1] = key;

}

}

void selectionsort(int arr[], int n)

{

int i, min, j, temp;

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

{

min = i;

for (j = i + 1; j < n; j++)

{

if (arr[min] > arr[j])

{

min = j;

}

}

temp = arr[min];

arr[min] = arr[i];

arr[i] = temp;

}

}

int main()

{

int num = 50000;

int r, arr[num], arr2[num];

srand(time(NULL));

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

{

r = rand() % num;

arr[i] = r;

}

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

{

arr2[i] = arr[i];

}

// avg case

clock\_t begin1 = clock();

selectionsort(arr, num);

clock\_t end1 = clock();

double time\_spent1 = (double)(end1 - begin1) / CLOCKS\_PER\_SEC;

clock\_t begin2 = clock();

insertionSort(arr2, num);

clock\_t end2 = clock();

double time\_spent2 = (double)(end2 - begin2) / CLOCKS\_PER\_SEC;

printf("Average Case: Time taken by Insertion Sort:%f \n\tTime taken by Selection Sort:%f ", time\_spent2, time\_spent1);

// best case

begin1 = clock();

selectionsort(arr, num);

end1 = clock();

time\_spent1 = (double)(end1 - begin1) / CLOCKS\_PER\_SEC;

begin2 = clock();

insertionSort(arr2, num);

end2 = clock();

time\_spent2 = (double)(end2 - begin2) / CLOCKS\_PER\_SEC;

printf("\nBest Case: Time taken by Insertion Sort:%f \n\tTime taken by Selection Sort:%f ", time\_spent2, time\_spent1);

// worst case here

int temp, j = num - 1;

for (int i = 0; i < num / 2; i++)

{

temp = arr[j];

arr[j] = arr[i];

arr[i] = temp;

temp = arr2[j];

arr2[j] = arr2[i];

arr2[i] = temp;

j = j - 1;

}

begin1 = clock();

selectionsort(arr, num);

end1 = clock();

time\_spent1 = (double)(end1 - begin1) / CLOCKS\_PER\_SEC;

begin2 = clock();

insertionSort(arr2, num);

end2 = clock();

time\_spent2 = (double)(end2 - begin2) / CLOCKS\_PER\_SEC;

printf("\nWorst Case: Time taken by Insertion Sort:%f \n\tTime taken by Selection Sort:%f ", time\_spent2, time\_spent1);

return 0;

}

**2. Write a program to implement and analyse time complexity of Merge sort, Quick Sort**

Merge Sort

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

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

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[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 num = 50000;

int arr[num], r;

srand(time(NULL));

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

{

r = rand() % num;

arr[i] = r;

}

// avg case

clock\_t begin1 = clock();

mergeSort(arr, 0, num - 1);

clock\_t end1 = clock();

double time\_spent1 = (double)(end1 - begin1) / CLOCKS\_PER\_SEC;

printf("Average Case (Random Order): Time taken by Merge Sort:%f \n", time\_spent1);

begin1 = clock();

mergeSort(arr, 0, num - 1);

end1 = clock();

time\_spent1 = (double)(end1 - begin1) / CLOCKS\_PER\_SEC;

printf("Sorted Array Case (Increasing Order): Time taken by Merge Sort:%f \n", time\_spent1);

int temp, j = num - 1;

for (int i = 0; i < num / 2; i++)

{

temp = arr[j];

arr[j] = arr[i];

arr[i] = temp;

j = j - 1;

}

clock\_t begin2 = clock();

mergeSort(arr, 0, num - 1);

clock\_t end2 = clock();

double time\_spent2 = (double)(end2 - begin2) / CLOCKS\_PER\_SEC;

printf("Sorted Array Case (Decresing Order): Time taken by Merge Sort:%f \n", time\_spent2);

return 0;

}

Quicksort->

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

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

{

int t = \*a;

\*a = \*b;

\*b = t;

}

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 num = 50000;

int arr[num], r;

srand(time(NULL));

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

{

r = rand() % num;

arr[i] = r;

}

// avg case

// printf("\nArray-\n");

// for(int i=0;i<num;i++){

// printf("%d ",arr[i]);

// }

clock\_t begin1 = clock();

quickSort(arr, 0, num - 1);

clock\_t end1 = clock();

double time\_spent1 = (double)(end1 - begin1) / CLOCKS\_PER\_SEC;

printf("\nAverage Case (Random Order): Time taken by Quick Sort:%f \n", time\_spent1);

// Best case

begin1 = clock();

quickSort(arr, 0, num - 1);

end1 = clock();

time\_spent1 = (double)(end1 - begin1) / CLOCKS\_PER\_SEC;

printf("\nSorted Array Case (Increasing Order):: Time taken by Quick Sort:%f \n", time\_spent1);

// Worst Case

int temp, j = num - 1;

for (int i = 0; i < num / 2; i++)

{

temp = arr[j];

arr[j] = arr[i];

arr[i] = temp;

j = j - 1;

}

clock\_t begin2 = clock();

quickSort(arr, 0, num - 1);

clock\_t end2 = clock();

double time\_spent2 = (double)(end2 - begin2) / CLOCKS\_PER\_SEC;

printf("\nSorted Array Case (Decresing Order):: Time taken by Quick Sort:%f \n", time\_spent2);

return 0;

}

**3. Write a program to implement Single source shortest path using Dynamic Programming**

BellmanFord-

#include<stdio.h>

int graph[20][20];

int distance[20];

int num = 20;

void InitialiseSingleSource(int source)

{

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

{

distance[i] = 9999;

}

distance[source] = 0;

}

void relax(int u, int v)

{

if (distance[v] > distance[u] + graph[u][v])

{

distance[v] = distance[u] + graph[u][v];

}

}

int BellmanFord(int source)

{

InitialiseSingleSource(source);

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

{

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

{

for (int k = 0; k < num; k++)

{

if (graph[j][k] == 0)

continue;

relax(j, k);

}

}

}

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

{

for (int k = 0; k < num; k++)

{

if (graph[j][k] == 0)

continue;

if (distance[k] > distance[j] + graph[j][k])

{

printf("Graph has a Negative Cycle");

return 0;

}

}

}

return 1;

}

int main()

{

int ans = 1;

printf("Enter Number of Vertices: ");

scanf("%d", &num);

printf("Enter Weight edges : \n");

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

{

printf("From Vertex %d: ", i + 1);

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

{

scanf("%d", &graph[i][j]);

}

}

ans = BellmanFord(0);

if (ans)

{

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

{

printf("%d ", distance[i]);

}

}

return 0;

}

**4. Write a program to implement Longest common subsequence**

#include <stdio.h>

#include <string.h>

int b[25][25], c[25 + 1][25 + 1];

void LCS\_Length(char x[], char y[])

{

int m = strlen(x);

int n = strlen(y);

for (int i = 0; i < m + 1; i++)

{

c[0][i] = 0;

}

for (int i = 0; i < m + 1; i++)

{

c[i][0] = 0;

}

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

{

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

{

if (x[i] == y[j])

{

c[i + 1][j + 1] = c[i][j] + 1;

b[i][j] = 0;

}

else if (c[i][j + 1] >= c[i + 1][j])

{

c[i + 1][j + 1] = c[i][j + 1];

b[i][j] = 1;

}

else

{

c[i + 1][j + 1] = c[i + 1][j];

b[i][j] = 2;

}

}

}

return;

}

void Print\_LCS(char x[], int i, int j)

{

if (i == -1 || j == -1)

return;

if (b[i][j] == 0)

{

Print\_LCS(x, i - 1, j - 1);

printf("%c ", x[i]);

}

else if (b[i][j] == 1)

{

Print\_LCS(x, i - 1, j);

}

else

{

Print\_LCS(x, i, j - 1);

}

}

int main()

{

char str1[100],str2[100];

printf("Enter String 1 : ");

scanf("%s",&str1);

printf("Enter String 2 : ");

scanf("%s",&str2);

LCS\_Length(str1, str2);

int m = strlen(str1);

int n = strlen(str2);

Print\_LCS(str1, m, n);

return 0;

}

**5. Write a program to implement Minimum Spanning Tree using Prim's and Kruskal Algorithm**

*Prims:-*

#include <stdio.h>

#include <stdlib.h>

int distances[100];

int completed[100];

int parent[100];

int ver;

int graph[100][100];

void initializeSource()

{

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

{

distances[i] = 100;

completed[i] = 0;

parent[i] = -1;

}

distances[0] = 0;

parent[0] = 0;

}

void relax(int u, int v)

{

if (distances[v] > graph[u][v])

{

distances[v] = graph[u][v];

parent[v] = u;

}

}

int extractMin()

{

int min;

int some = 0;

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

{

if (completed[i] != 1)

{

some = 1;

min = i;

break;

}

}

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

{

if (completed[i] == 0 && distances[i] <= distances[min])

{

min = i;

}

}

return min;

}

int main()

{

printf("Enter Total Number Of Vertices In Graph: ");

scanf("%d",&ver);

printf("Enter Distances For each Vertex-\n");

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

{

printf("From Vertex %c : ",i + 97);

for(int j=0;j<ver;j++){

scanf("%d",&graph[i][j]);

}

}

initializeSource();

int cur;

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

{

cur = extractMin();

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

{

if (completed[i] == 0 && graph[cur][i] != 0)

{

relax(cur, i);

}

}

completed[cur] = 1;

}

int sum = 0;

printf("Distances\t");

printf("Parent\n");

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

{

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

printf(" \t%c\n", parent[i] + 97);

sum = sum + distances[i];

}

printf("Cost Of Minimum Spanning Tree: %d\n", sum);

return 0;

}

*Kruskal:-*

#include <stdio.h>

int graph[100][100];

int set[100];

int parent[100];

int ver;

int edge\_no = 0;

struct edge

{

int pt1;

int pt2;

int wt;

int sol;

};

struct edge edges[100];

void edgesCalulate()

{

int count = 0;

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

{

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

{

if (graph[i][j] != 0)

{

edges[count].pt1 = i;

edges[count].pt2 = j;

edges[count].wt = graph[i][j];

edges[count].sol = 0;

count++;

}

}

}

edge\_no = count;

}

void edgesSort()

{

struct edge temp;

for (int i = 0; i < edge\_no; i++)

{

for (int j = 0; j < edge\_no - 1; j++)

{

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

{

temp = edges[j];

edges[j] = edges[j + 1];

edges[j + 1] = temp;

}

}

}

}

int main()

{

printf("Enter Total Number Of Vertices In Graph: ");

scanf("%d",&ver);

printf("Enter Distances For each Vertex-\n");

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

{

printf("From Vertex %c : ",i + 97);

for(int j=0;j<ver;j++){

scanf("%d",&graph[i][j]);

}

}

edgesCalulate();

edgesSort();

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

{

set[i] = i;

}

for (int i = 0; i < edge\_no; i++)

{

if (set[edges[i].pt1] != set[edges[i].pt2])

{

edges[i].sol = 1;

int temp;

temp = set[edges[i].pt1];

for (int j = 0; j < edge\_no; j++)

{

if (set[j] == temp)

set[j] = set[edges[i].pt2];

}

}

}

int sum = 0;

for (int i = 0; i < edge\_no; i++)

{

if (edges[i].sol)

{

sum = sum + edges[i].wt;

printf(" wt- %d ", edges[i].wt);

printf(" ( %c , %c ) \n", edges[i].pt1 + 97, edges[i].pt2 + 97);

}

}

printf("Minimum Cost Of Spanning Tree: %d ", sum);

return 0;

}

**6. Write a program to implement Single source shortest path using Greedy Approach**

#include <stdio.h>

#include <stdlib.h>

int graph[100][100];

int n;

int distances[100];

int completed[100];

int pi[100];

void initializeSource(){

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

distances[i]=100;

completed[i]=0;

pi[i]=-1;

}

distances[0]=0;

pi[0]=0;

}

void relax(int u,int v){

if(distances[v]>distances[u]+graph[u][v]){

distances[v]=distances[u]+graph[u][v];

pi[v]=u;

}

}

int extractMin(){

int min;

int some=0;

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

if(completed[i]!=1){

some=1;

min=i;

break;

}

}

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

if(completed[i]==0 && distances[i]<=distances[min]){

min=i;

}

}

return min;

}

int main(){

printf("Enter Number of Vertices: ");

scanf("%d",&n);

printf("Enter Weight of edges to all vertices : \n");

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

printf("Form Vertex %d: ",i);

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

scanf("%d",&graph[i][j]);

}

}

initializeSource();

int cur;

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

cur = extractMin();

for(int i=0;i<5;i++){

if(completed[i]==0 && graph[cur][i]!=0){

relax(cur,i);

}

}

completed[cur]=1;

}

printf("Vertex Distance Parent \n");

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

printf("%d %d %d\n",i, distances[i],pi[i]);

}

return 0;

}

**7. Write a program to implement n queens Problem.**

#include<stdio.h>

#include<math.h>

int n;

int x[20];

int Place(int k,int i){

int j;

for (j=1;j<=k-1;j++){

if(x[j]==i || abs(x[j]-i)==abs(j-k)){

return 0;

}

}

return 1;

}

void N\_Queens(int k){

int i,j;

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

if(Place(k,i)==1){

x[k]=i;

if(k==n){

printf("Answer: ");

for(j=1;j<=n;j++){

printf(" %d ",x[j]);

}

printf("\n");

return;

}

else{

N\_Queens(k+1);

}

}

}

}

int main(){

int l;

printf("Enter n: ");

scanf("%d",&n);

N\_Queens(1);

return 0;

}

**8. Write a program to implement sum of Subsets**

#include<stdio.h>

int counter=1;

int n;

int arr[50];

int ans[50];

int m;

int total=0;

void sumOfSubset(int s, int k, int r){

int j;

ans[k]=1;

if(s+arr[k]==m){

printf("Solution %d: ",counter,s);

counter++;

for(j=0;j<n;j++){

printf(" %d ",ans[j]);

}

printf("\n");

for(j=k;j<n;j++){

ans[j]=0;

}

return;

}

else if(k!=n-1 && s+arr[k]+arr[k+1]<=m){

sumOfSubset(s+arr[k],k+1,r-arr[k] );

}

if(s+r-arr[k]>=m && s+arr[k+1]<=m && k!=n-1 ){

ans[k]=0;

sumOfSubset(s,k+1,r-arr[k]);

}

}

int main(){

int i;

printf("Enter Total Number of Numbers to be used : ");

scanf("%d",&n);

printf("Enter Sum to be Calculated : ");

scanf("%d",&m);

printf("Enter Numbers : ");

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

{

scanf("%d",&arr[i]);

ans[i]=0;

}

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

total+=arr[i];

}

sumOfSubset(0,0,total);

return 0;

}

**9. Write a program to implement Graph Colouring**

#include <stdio.h>

int graph[100][100];

int m;

int n;

int x[100];

void NextVal(int k){

int j=0;

while(1){

x[k]=(x[k]+1)%(m+1);

if(x[k]==0)

return;

for ( j = 0; j <n; j++) {

// printf("1- %d\n",x[j]);

if(graph[k][j]!=0 && x[k]==x[j]){

break;

}

}

if(j==n)

return;

}

}

void mColoring(int k){

while(1){

NextVal(k);

if(x[k]==0)

return;

if(k==n-1){

printf("Solution is-\n");

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

printf(" %d ",x[i]);

}

printf("\n");

break;

}

else{

mColoring(k+1);

}

}

}

int main()

{

printf("Enter Number of Vertices: ");

scanf("%d",&n);

printf("Enter Distances For each Vertex-\n");

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

{

printf("From Vertex %c : ",i + 97);

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

scanf("%d",&graph[i][j]);

}

x[i]=0;

}

printf("Enter Number of Colors: ");

scanf("%d",&m);

mColoring(0);

return 0;

}

**10. Write a program to implement Rabin Karp String matching Algorithm and KMP algorithm**

**Rabin Karp-**

#include<stdio.h>

#include<string.h>

#include<math.h>

void rabinK(char txt[],char pat[],int d,int q ){

int i,s,m,n,h=1,hashP=0,hashT=0;

n=strlen(txt);

m=strlen(pat);

h= ( int )(pow(d,m-1))%q;

for( i=0;i<m;i++){

hashP=(d\*hashP + pat[i])%q;

hashT=(d\*hashT +txt[i])%q;

}

for( s=0; s<=n-m;s++){

if(hashP==hashT){

for(i=0;i<m;i++){

if(pat[i]!=txt[s+i]){

break;

}

}

if(i==m){

printf("Pattern Occured at index %d. \n",s);

}

}

if(s<n-m){

hashT=(d\*(hashT-(txt[s]\*h))+txt[s+m])%q;

if(hashT<0){

hashT+=q;

}

}

}

return;

}

int main(){

int d,q;

char txt[ 100],pat[100] ;

printf("Enter Text: ");

scanf("%s",&txt);

printf("Enter Pattern: ");

scanf("%s",&pat);

d=256;

q=31;

rabinK(txt,pat,d,q);

return 0;

}

**KMP-**

#include<stdio.h>

#include<string.h>

int pi[100];

void computePrefix(char pat[]){

int q,m,k=0;

m=strlen(pat);

pi[0]=0;

for(q=1;q<m;q++){

while(k>0 && pat[k]!=pat[q]){

k=pi[k-1];

}

if(pat[k]==pat[q]){

k=k+1;

}

pi[q]=k;

}

return;

}

void kmp(char txt[],char pat[]){

int i,m,n,q;

n=strlen(txt);

m=strlen(pat);

computePrefix(pat);

q=0;

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

while(q>0 && pat[q]!=txt[i]){

q=pi[q-1];

}

if (pat[q]==txt[i])

q=q+1;

if (q==m){

printf("Pattern Occurs at index %d \n",(i-(m-1)));

q=pi[q-1];

}

}

return;

}

int main(){

char txt[ 100],pat[100 ] ;

printf("Enter Text: ");

scanf("%s",&txt);

printf("Enter Pattern: ");

scanf("%s",&pat);

kmp(txt,pat);

return 0;

}