**EXPERIMENT NO: 1**

**AIM:-** To implement the selection sort algorithms and analyze their running time.

**CODE:-**

#include <stdio.h>

void selectionSort(int a[], int n){

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

int mi= i;

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

if (a[j] < a[mi]){

mi= j;

}

}

if(mi!= i){

int temp=a[mi];

a[mi]=a[i];

a[i]=temp;

}

}

}

int main(){

int a[] = {64, 25, 12, 22, 11};

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

selectionSort(a, n);

printf("Sorted array:");

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

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

}

return 0;

}

**OUTPUT:-** Sorted array:11 12 22 25 64

**EXPERIMENT NO: 2**

**AIM:-** To implement the insertion sort algorithms and analyze their running time.

**CODE:-**

#include <stdio.h>

void insertionSort(int arr[], int n){

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

int key = arr[i];

int j = i - 1;

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

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

j--;

}

arr[j + 1] = key;

}

}

int main(){

int a[] = {12, 11, 13, 5, 6};

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

insertionSort(a, n);

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

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

}

return 0;

}

**OUTPUT:-** 5 6 11 12 13

**EXPERIMENT NO: 3**

**AIM:-** To implement the bubble sort algorithms and analyze their running time.

**CODE:-**

#include <stdio.h>

void bubbleSort(int arr[], int n){

int i, j;

bool swapped;

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

swapped = false;

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

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

int temp=a[j];

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

a[j+1]=temp;

swapped = true;

}

}

if (swapped == false)

break;

}

}

int main(){

int arr[] = { 64, 34, 25, 12, 22, 11, 90 };

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

bubbleSort(arr, n);

printf("Sorted array:");

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

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

}

return 0;

}

**OUTPUT:-** 5 6 11 12 13

**EXPERIMENT NO:- 4**

**Aim:-** To implement the concept of divide and conquer strategy of designing algorithms for the Merge sorting algorithm and analyze their running time.

**CODE:-**

#include <stdio.h>

void merge(int a[], 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] = a[l + i];

}

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

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

}

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

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

a[k] = L[i];

i++;

}

else {

a[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

a[k] = L[i];

i++;

k++;

}

while (j < n2) {

a[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 a[] = { 12, 11, 13, 5, 6, 7 };

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

mergeSort(a, 0, n - 1);

printf("Sorted array is:");

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

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

}

return 0;

}

**OUTPUT:-** Sorted array is:5 6 7 11 12 13

**EXPERIMENT NO:- 5**

**AIM:-** To implement the concept of divide and conquer strategy of designing algorithms for the Quick sort algorithm and analyze their running time.

**CODE:-**

#include <stdio.h>

void swap(int \*x,int \*y){

int temp=\*x;

\*x=\*y;

\*y=temp;

}

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

int pivot=arr[high];

int i=(low-1);

for(int j=low;j<=high;j++){

if(arr[j]<pivot){

i++;

swap(&arr[i],&arr[j]);

}

}

swap(&arr[i+1],&arr[high]);

return (i+1);

}

void quickSort(int arr[],int low,int high){

if(low<high){

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

quickSort(arr,low,pi-1);

quickSort(arr,pi+1,high);

}

}

int main() {

int arr[]={10,7,8,9,1,5};

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

quickSort(arr,0,n-1);

printf("Sorted Array:");

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

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

}

return 0;

}

**OUTPUT:-** Sorted Array:1 5 7 8 9 10

**EXPERIMENT NO:6**

**AIM:-** To implement the linear search algorithm.

**CODE:-**

#include <stdio.h>

int search(int arr[], int N, int x)

{

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

if (arr[i] == x)

return i;

return -1;

}

int main() {

int a[] = { 2, 3, 4, 10, 40 };

int x = 10;

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

int result = search(a, n, x);

(result == -1)

? printf("Element is not present in array")

: printf("Element is present at index %d", result);

return 0;

}

**OUTPUT:-** Element is present at index 3

ii.Binary search

**EXPERIMENT NO:7**

**AIM:-** To implement the binary search algorithm.

**CODE:-**

#include <stdio.h>

int binarySearch(int arr[], int l, int r, int x){

while (l <= r) {

int m = l + (r - l) / 2;

if (arr[m] == x)

return m;

if (arr[m] < x)

l = m + 1;

else

r = m - 1;

}

return -1;

}

int main(){

int a[] = { 2, 3, 4, 10, 40 };

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

int x = 10;

int res= binarySearch(a, 0, n - 1, x);

(res== -1) ? printf("Element is not present in array")

: printf("Element is present at index %d",res);

return 0;

}

**OUTPUT:-**Element is present at index 3

**EXPERIMENT NO:8**

**AIM:-** To implement Counting Sort linear time algorithm and analyze its running time.

**CODE:-**

#include<stdio.h>

void countSort(int a[],int o[],int n){

int m=0;

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

if(m<a[i]){

m=a[i];

}

}

int c[m+1];

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

c[i]=0;

}

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

c[a[i]]++;

}

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

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

}

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

o[c[a[i]] - 1]=a[i];

c[a[i]]--;

}

}

int main(){

int a[]= { 4, 3, 12, 1, 5, 5, 3, 9 };

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

int o[n];

countSort(a,o,n);

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

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

}

return 0;

}

**OUTPUT:-**

1 3 3 4 5 5 9 12

**EXPERIMENT NO: 9**

**AIM:-** To implement the Strassen Matrix Multiplication using greedy programming.

**CODE:-**

#include <bits/stdc++.h>

using namespace std;

#define ROW\_1 4

#define COL\_1 4

#define ROW\_2 4

#define COL\_2 4

void print(string display, vector<vector<int> > matrix,int start\_row,

int start\_column, int end\_row,int end\_column){

cout << display << " =>" << endl;

for (int i = start\_row; i <= end\_row; i++) {

for (int j = start\_column; j <= end\_column; j++) {

cout << setw(10);

cout << matrix[i][j];

}

cout << endl;

}

cout << endl;

return;

}

vector<vector<int> >add\_matrix(vector<vector<int> > matrix\_A,

vector<vector<int> > matrix\_B, int split\_index,int multiplier = 1){

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

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

matrix\_A[i][j]= matrix\_A[i][j]+ (multiplier \* matrix\_B[i][j]);

return matrix\_A;

}

vector<vector<int> >

multiply\_matrix(vector<vector<int> > matrix\_A,vector<vector<int> > matrix\_B){

int col\_1 = matrix\_A[0].size();

int row\_1 = matrix\_A.size();

int col\_2 = matrix\_B[0].size();

int row\_2 = matrix\_B.size();

if (col\_1 != row\_2) {

cout << "\nError: The number of columns in A must be equal to the number of rows in B\n";

return {};

}

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

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

if (col\_1 == 1)

result\_matrix[0][0]= matrix\_A[0][0] \* matrix\_B[0][0];

else {

int split\_index = col\_1 / 2;

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

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);

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

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

a00[i][j] = matrix\_A[i][j];

a01[i][j] = matrix\_A[i][j + split\_index];

a10[i][j] = matrix\_A[split\_index + i][j];

a11[i][j] = matrix\_A[i + split\_index][j + split\_index];

b00[i][j] = matrix\_B[i][j];

b01[i][j] = matrix\_B[i][j + split\_index];

b10[i][j] = matrix\_B[split\_index + i][j];

b11[i][j] = matrix\_B[i + split\_index][j + split\_index];

}

vector<vector<int> > p(multiply\_matrix(a00, add\_matrix(b01, b11, split\_index, -1)));

vector<vector<int> > q(multiply\_matrix(add\_matrix(a00, a01, split\_index), b11));

vector<vector<int> > r(multiply\_matrix(add\_matrix(a10, a11, split\_index), b00));

vector<vector<int> > s(multiply\_matrix(a11, add\_matrix(b10, b00, split\_index, -1)));

vector<vector<int> > t(multiply\_matrix(

add\_matrix(a00, a11, split\_index),add\_matrix(b00, b11, split\_index)));

vector<vector<int> > u(multiply\_matrix(

add\_matrix(a01, a11, split\_index, -1),add\_matrix(b10, b11, split\_index)));

vector<vector<int> > v(multiply\_matrix(

add\_matrix(a00, a10, split\_index, -1),add\_matrix(b00, b01, split\_index)));

vector<vector<int> > result\_matrix\_00(add\_matrix(add\_matrix(add\_matrix(t, s, split\_index), u,split\_index),q, split\_index, -1));

vector<vector<int> > result\_matrix\_01(add\_matrix(p, q, split\_index));

vector<vector<int> > result\_matrix\_10(add\_matrix(r, s, split\_index));

vector<vector<int> > result\_matrix\_11(add\_matrix(add\_matrix(add\_matrix(t, p, split\_index), r,split\_index, -1),v, split\_index, -1));

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

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

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

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

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

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

}

a00.clear(),a01.clear(),a10.clear(),a11.clear();

b00.clear(),b01.clear(),b10.clear(),b11.clear();

p.clear(),q.clear(), r.clear(),s.clear();

t.clear(),u.clear(),v.clear();

result\_matrix\_00.clear(),result\_matrix\_01.clear();

result\_matrix\_10.clear(),result\_matrix\_11.clear();

}

return result\_matrix;

}

int main(){

vector<vector<int> > matrix\_A = { { 1, 1, 1, 1 },{ 2, 2, 2, 2 },{ 3, 3, 3, 3 },{ 2, 2, 2, 2 } };

print("Array A", matrix\_A, 0, 0, ROW\_1 - 1, COL\_1 - 1);

vector<vector<int> > matrix\_B = { { 1, 1, 1, 1 },{ 2, 2, 2, 2 },{ 3, 3, 3, 3 },{ 2, 2, 2, 2 } };

print("Array B", matrix\_B, 0, 0, ROW\_2 - 1, COL\_2 - 1);

vector<vector<int> > result\_matrix(multiply\_matrix(matrix\_A, matrix\_B));

print("Result Array", result\_matrix, 0, 0, ROW\_1 - 1,COL\_2 - 1);

}

**OUTPUT:-**

Array A =>

1 1 1 1

2 2 2 2

3 3 3 3

2 2 2 2

Array B =>

1 1 1 1

2 2 2 2

3 3 3 3

2 2 2 2

Result Array =>

8 8 8 8

16 16 16 16

24 24 24 24

16 16 16 16

**EXPERIMENT NO:10**

**AIM:-** To implement a fractional Knapsack algorithm using greedy programming.

**CODE:-**

#include <bits/stdc++.h>

using namespace std;

struct Item {

int profit, weight;

Item(int profit, int weight){

this->profit = profit;

this->weight = weight;

}

};

static bool cmp(struct Item a, struct Item b){

double r1 = (double)a.profit / (double)a.weight;

double r2 = (double)b.profit / (double)b.weight;

return r1 > r2;

}

double fractionalKnapsack(int W, struct Item arr[], int N){

sort(arr, arr + N, cmp);

double finalvalue = 0.0;

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

if (arr[i].weight <= W) {

W -= arr[i].weight;

finalvalue += arr[i].profit;

}

else {

finalvalue+= arr[i].profit\* ((double)W / (double)arr[i].weight);

break;

}

}

return finalvalue;

}

int main(){

int W = 45;

Item arr[] = { { 60, 10 }, { 100, 20 }, { 120, 30 } };

int N = sizeof(arr) / sizeof(arr[0]);

cout << fractionalKnapsack(W, arr, N);

return 0;

}

**OUTPUT:-** 220

**EXPERIMENT NO:11**

**AIM:-** Implement the prim’s algorithm to find the MST.

**CODE:-**

#include<bits/stdc++.h>

using namespace std;

typedef pair<int,int> pii;

int MST(int V, int E, int edges[][3]){

vector<vector<int>> adj[V];

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

int u = edges[i][0];

int v = edges[i][1];

int wt = edges[i][2];

adj[u].push\_back({v, wt});

adj[v].push\_back({u, wt});

}

priority\_queue<pii, vector<pii>, greater<pii>> pq;

vector<bool> vis(V, false);

int ans= 0;

pq.push({0, 0});

while(!pq.empty()){

auto p = pq.top();

pq.pop();

int wt = p.first;

int u = p.second;

if(vis[u] == true){

continue;

}

ans+= wt;

vis[u] = true;

for(auto v : adj[u]){

if(vis[v[0]] == false){

pq.push({v[1], v[0]});

}

}

}

return ans;

}

int main(){

int graph[][3] = {{0, 1, 5},{1, 2, 3},{0, 2, 1}};

cout << mst(3, 3, graph) << endl;

return 0;

}

**OUTPUT:-** 4

**EXPERIMENT NO:12**

**AIM:-** Implement the kruskal's algorithm to find the MST.

**CODE:-** #include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

#define edge pair<int, int>

class Graph {

private:

vector<pair<int, edge> > G;

vector<pair<int, edge> > T;

int \*parent;

int V;

public:

Graph(int V);

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

int findSet(int i);

void unionSet(int u, int v);

void kruskal();

void print();

};

Graph::Graph(int V) {

parent = new int[V];

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

parent[i] = i;

G.clear();

T.clear();

}

void Graph::AddWE(int u, int v, int w) {

G.push\_back(make\_pair(w, edge(u, v)));

}

int Graph::findSet(int i) {

if (i == parent[i])

return i;

else

return findSet(parent[i]);

}

void Graph::unionSet(int u, int v) {

parent[u] = parent[v];

}

void Graph::kruskal() {

int i, uRep, vRep;

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

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

uRep = findSet(G[i].second.first);

vRep = findSet(G[i].second.second);

if (uRep != vRep) {

T.push\_back(G[i]);

unionSet(uRep, vRep);

}

}

}

void Graph::print() {

int c=0;

cout << "Edge :"

<< " Weight" << endl;

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

cout << T[i].second.first << " - " << T[i].second.second << " : "

<< T[i].first<<endl;

c+=T[i].first;

}

cout<<"Cost:"<<c<<endl;

}

int main() {

Graph g(6);

g.AddWE(0, 1, 4), g.AddWE(0, 2, 4);

g.AddWE(1, 2, 2), g.AddWE(1, 0, 4);

g.AddWE(2, 0, 4);

g.AddWE(2, 1, 2);

g.AddWE(2, 3, 3);

g.AddWE(2, 5, 2);

g.AddWE(2, 4, 4);

g.AddWE(3, 2, 3);

g.AddWE(3, 4, 3);

g.AddWE(4, 2, 4);

g.AddWE(4, 3, 3);

g.AddWE(5, 2, 2);

g.AddWE(5, 4, 3);

g.kruskal();

g.print();

return 0;

}

**OUTPUT:-**

Edge : Weight

1 - 2 : 2

2 - 5 : 2

2 - 3 : 3

3 - 4 : 3

0 - 1 : 4

Cost:14

**EXPERIMENT NO:13**

**AIM:-** To Implement algorithm using Dynamic programming approach to solve Longest common subsequence.

**CODE:-**

#include <bits/stdc++.h>

using namespace std;

int lcs(string X, string 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));

}

int main(){

string S1 = "AGGTAB";

string S2 = "GXTXAYB";

int m = S1.size();

int n = S2.size();

cout << "Length of LCS is " << lcs(S1, S2, m, n);

return 0;

}

**OUTPUT:-**

Length of LCS is 4