# Find square root of a number. Can we use Divide & Conquer approach for this problem? Can we have a still better algorithm to solve the problem?

**CODE:**

#include<stdio.h> #include<math.h> void srqroot(int n); void main()

{

int n;

printf("\n Enter number :-"); scanf("%d",&n);

srqroot(n);

}

void srqroot(int n)

{

}

# OUTPUT:

int ans;

ans = sqrt(n);

printf("\n Square root of %d is = %d",n,ans);

Enter number :-25 Square root of 25 is = 5

**OR**

# CODE:

#include<stdio.h> void root(int n)

{

int i = 0; double root=1; while(1)

{

i = i+1;

root = (n/root+root)/2; if(i == n+1)

{

break;

}

}

printf("Square root of %d = %2.f",n,root);

}

void main()

{

int n;

printf("\n Enter number : "); scanf("%d",&n);

root(n);

}

# OUTPUT:

Enter number : 25 Square root of 25 = 5

# Determine smallest divisor of an integer.

**CODE:**

#include<stdio.h> #include<math.h>

int smallest\_divisior(int n){ int i;

for(i = 2;i<=sqrt(n);i++)

{

if(n%i == 0)

{

return i;

}

}

return n;

}

void main()

{

int n,sd;

printf("\n Enter number : "); scanf("%d",&n);

sd = smallest\_divisior(n);

printf("\n Smallest divisor of %d = %d ",n,sd);

}

# OUTPUT:

Enter number : 16 Smallest divisor of 16 = 2

# OR

**CODE:**

#include<stdio.h>

void smallest\_divisor(int n)

{

int i;

if(n == 1)

{

printf("\n\n The smallest divisor of %d is %d",n,n);

}

i = 2;

while(i <= n)

{

if(n % i == 0)

{

printf("\n\n The smallest divisor of %d is %d\n",n,i); break;

} i++;

}

}

void main()

{

int n;

printf("\n\n Enter number : "); scanf("%d", &n); smallest\_divisor(n);

}

# OUTPUT:

Enter number : 81

The smallest divisor of 81 is 3

# For a given value of n, generate prime numbers <= n (more than one algorithms are possible).

**CODE:**

#include<stdio.h>

void prime(int n)

{

int i,j;

printf("Prime numbers is : "); for(i=1;i<=n;i++)

{

for(j=2;j<n;j++)

{

if(i%j==0)

{

break;

}

}

if(j==i)

{

printf("\t%d",i);

}

}

}

void main()

{

int n;

printf("Enter Number: "); scanf("%d",&n); prime(n);

}

# OUTPUT:

Enter Number: 13

Prime numbers is : 2 3 5 7 11 13

# OR

**CODE:**

#include<stdio.h>

void prime(int n)

{

int i,j,count=0;

printf("\nPrime numbers are :\n "); for(i=0;i<=n;i++)

{

count = 0; for(j=1;j<=i;j++)

{

if(i%j == 0)

{

count++;

}

}

if(count == 2)

{

printf("%d\n",i);

}

}

}

void main()

{

int n;

printf("\n Enter number : "); scanf("%d",&n);

prime(n);

}

# OUTPUT:

Enter number : 15 Prime numbers are : 2

3

5

7

11

13

# Find Xn. Iterative and recursive algorithms are possible with complexity log 2 n.

**CODE:**

#include<stdio.h> void power(int n,int p)

{

int i,power=1; for(i=1;i<=p;i++)

{

power = power \* n;

}

printf("\n %d^%d = %d",n,p,power);

}

void main()

{

int n,p;

printf("\n Enter x : "); scanf("%d",&n);

printf("\n Enter power to be find : "); scanf("%d",&p);

power(n,p);

}

# OUTPUT:

Enter x : 8

Enter power to be find : 3\ 8^3 = 512

# OR

**CODE:**

#include<stdio.h> int power(int n,int p)

{

if(p == 1)

{

}

else

{

}

}

return n;

return(n\*power(n,p-1));

void main()

{

int n,p,pwr;

printf("\n Enter number : "); scanf("%d",&n);

printf("\n Enter power to be find : ");

}

# OUTPUT:

scanf("%d",&p); pwr = power(n,p);

printf("\n value of %d^%d is = %d",n,p,pwr);

Enter number : 4

Enter power to be find : 2 value of 4^2 is = 16

# Determine product of 2 integers (a \* b) as repeated sums. Iterative and recursive algorithms are possible.

**CODE:**

#include<stdio.h>

void repeated\_sum(int a,int b)

{

int sum = 0,i; for(i=0;i<b;i++)

{

sum = sum + a;

}

printf("product of two integet %d and %d using repeated sum = %d",a,b,sum);

}

void main()

{

int a,b;

printf("\n Enter value of a :"); scanf("%d",&a);

printf("\n Enter value of b : "); scanf("%d",&b); repeated\_sum(a,b);

}

# OUTPUT:

Enter value of a :5 Enter value of b : 6

product of two integet 5 and 6 using repeated sum = 30

# OR

**CODE:**

#include<stdio.h>

int repeat\_sum(int a,int b)

{

if(b == 1)

{

return a;

}

return(a+repeat\_sum(a,b-1)) ;

}

void main()

{

int a,b,product;

printf("\n Enter value of a :"); scanf("%d",&a);

printf("\n Enter value of b : "); scanf("%d",&b);

product = repeat\_sum(a,b);

printf("product of two integet %d and %d using repeated sum = %d",a,b,product);

}

# OUTPUT:

Enter value of a :6 Enter value of b : 6

product of two integet 6 and 6 using repeated sum = 36

# Find Factorial of n. Iterative and recursive algorithms are possible.

**CODE:**

#include<stdio.h> int main()

{

int i,fact=1,number; printf("Enter Number : "); scanf("%d",&number); for(i=1;i<=number;i++)

{

fact=fact\*i;

}

printf("\nFACTORIAL OF %d IS : %d",number,fact); return 0;

}

# OR

#include<stdio.h> long factorial(int n)

{

if (n == 0)

return 1;

else

return(n \* factorial(n-1));

}

void main()

{

int number; long fact;

printf("Enter Number: "); scanf("%d", &number); fact = factorial(number);

printf("\nFACTORIAL OF %d IS : %d", number, fact); return 0;

}

# OUTPUT:

Enter Number: FACTORIAL OF 6 IS: 720

# Generate Fibonacci series up to n terms Iterative and recursive algorithms are possible.

**CODE:**

#include<stdio.h> int main(){

int sum = 0, n,a = 0,b = 1; printf("Enter Number: "); scanf("%d", &n);

printf("\nFIBINACCI SERIES IS: \n\n"); while(sum <= n){

printf("%d\t ", sum); a = b;

b = sum; sum = a + b;

}

return 0;

}

# OR

#include<stdio.h> int Fibonacci(int n){

if (n <= 1) return n;

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

}

int main(int argc, char \*argv[]){ int n, sum = 0;

n = atoi(argv[1]); while(Fibonacci(sum) <= n){

printf("%d ", Fibonacci(sum)); sum++;

}

}

# OUTPUT:

Enter Number: 10 FIBINACCI SERIES IS:

0 1 1 2 3 5 8

# Determine product of 2 large integers using multiplication of their digits. For simplicity, assume both numbers to have same number of digits. This assumption can be relaxed subsequently.

**CODE:**

#include <stdio.h> int product(int, int); int main()

{

int a, b, result;

printf("ENTER TWO NUMBER TO FIND PRODUCT : ");

scanf("%d%d", &a, &b); result = product(a, b);

printf("\nPRODUCT OF %d AND %d IS %d\n", a, b, result); return 0;

}

int product(int a, int b)

{

if (a < b)

{

}

return product(b, a);

else if (b != 0)

{

return (a + product(a, b - 1));

}

else

{

return 0;

}

}

# OUTPUT:

ENTER TWO NUMBER TO FIND PRODUCT: 5 4 PRODUCT OF 5 AND 4 IS 20

# OR

**CODE:**

#include <stdio.h> #include <string.h> int main(){

int a[100],b[100],ans[200]={0},i,j,tmp; char s1[101],s2[101];

printf("Enter number 1 :"); scanf("%s",s1); printf("Enter number 2 :"); scanf("%s",s2);

int l1 = strlen(s1); int l2 = strlen(s2);

for(i = l1-1,j=0;i>=0;i--,j++){a[j] = s1[i]-'0'; }

for(i = l2-1,j=0;i>=0;i--,j++){b[j] = s2[i]-'0'; }

for(i = 0;i < l2;i++) { for(j = 0;j < l1;j++){

ans[i+j] += b[i]\*a[j];

}

}

for(i = 0;i < l1+l2;i++){ tmp = ans[i]/10; ans[i] = ans[i]%10;

ans[i+1] = ans[i+1] + tmp;

}

for(i = l1+l2; i>= 0;i--){ if(ans[i] > 0)

break;

}

printf("Product : ");

for(;i>= 0;i--)

{printf("%d",ans[i]); } return 0; }

# OUTPUT:

Enter number 1 :5

Enter number 2 :4

Product : 20

# Program for finding maximum and minimum number using Divide and conquer.

**CODE:**

#include <stdio.h> #define MAX\_SIZE 100 int main(){

int arr[MAX\_SIZE]; int i, max, min, size;

printf("ENTER SIZE OF THE ARRAY: ");

scanf("%d", &size);

printf("\nENTER %d ELEMENTS \n\n",size); for(i=0; i<size; i++){

printf("NUMBER %d : ",i+1); scanf("%d", &arr[i]);

}

max = arr[0]; min = arr[0];

for(i=1; i<size; i++){ if(arr[i] > max){

max = arr[i];

}

if(arr[i] < min){ min = arr[i];

}

}

printf("\nMAXIMUM ELEMENT IS : %d\n", max); printf("\nMINIMUM ELEMENT IS : %d", min); return 0;

}

# OUTPUT:

ENTER SIZE OF THE ARRAY: 5 ENTER 5 ELEMENTS

NUMBER 1: 55

NUMBER 2: 77

NUMBER 3: 22

NUMBER 4: 44

NUMBER 5: 11

MAXIMUM ELEMENT IS: 77 MINIMUM ELEMENT IS: 11

# Implement Recursive Binary search and Linear search and determine the time taken to search an element.

**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

}return -1;

}

r = m - 1;

int main(void){

int arr[100],n,i,x,result =0;

printf("ENTER THE NUMBER OF ELEMENT: ");

scanf("%d",&n); for(i=0;i<n;i++){

printf("\nELEMENT %d : ",i+1); scanf("%d",&arr[i]);

}

printf("\nENTER THE SEARCH ELEMENT: ");

scanf("%d",&x);

result = binarySearch(arr, 0, n - 1, x); (result == -1);

printf("\nELEMENT %d IS PRESENT AT INDEX %d",x,result); return 0;

}

# OUTPUT:

ENTER THE NUMBER OF ELEMENT: 5 ELEMENT 1: 2

ELEMENT 2: 4

ELEMENT 3: 3

ELEMENT 4: 6

ELEMENT 5: 5

ENTER THE SEARCH ELEMENT: 6 ELEMENT 6 IS PRESENT AT INDEX 3

# Breadth First Search (BFS) in a binary tree.

**CODE:**

#include <stdio.h> #include <stdlib.h> struct node {

int data;

struct node \*left, \*right;

};

void printCurrentLevel(struct node\* root, int level); int height(struct node\* node);

struct node\* newNode(int data);

void printLevelOrder(struct node\* root)

{

int h = height(root),i; for (i = 1; i<= h; i++)

printCurrentLevel(root, i);

}

void printCurrentLevel(struct node\* root, int level)

{

if (root == NULL)

return; if (level == 1)

printf("%d ", root->data); else if (level > 1) {

printCurrentLevel(root->left, level - 1); printCurrentLevel(root->right, level - 1);

}

}

int height(struct node\* node)

{

if (node == NULL)

return 0;

else {

int lheight = height(node->left); int rheight = height(node->right); if (lheight>rheight)

return (lheight + 1);

else

return (rheight + 1);

}

}

struct node\* newNode(int data)

{

struct node\* node = (struct node\*)malloc(sizeof(struct node)); node->data = data;

node->left = NULL; node->right = NULL; return (node);

}

int main()

{

struct node\* root = newNode(1); root->left = newNode(2);

root->right = newNode(3); root->left->left = newNode(4);

root->left->right = newNode(5);

printf("LEVEL ORDER TRAVERSAL OF BINARY SEARCH IS\n\n");

printLevelOrder(root); return 0;

}

# OUTPUT:

LEVEL ORDER TRAVERSAL OF BINARY SEARCH IS 1 2 3 4 5

# Depth First Search (DFS) in a binary tree.

**CODE:**

#include<stdio.h> #include<conio.h>

int a[20][20],reach[20],n; void dfs(int v)

{

int i; reach[v]=1;

for (i=1;i<=n;i++) if(a[v][i] && !reach[i]) {

printf("\n %d->%d",v,i); dfs(i);}

}

int main()

{

int i,j,count=0;

printf("\nENTER NUMBER OF VERTICES:");

scanf("%d",&n); for (i=1;i<=n;i++) {

reach[i]=0;

for (j=1;j<=n;j++) a[i][j]=0;

}

printf("\nENTER THE ADJACENCY MATRIX\n");

for (i=1;i<=n;i++) for (j=1;j<=n;j++) scanf("%d",&a[i][j]); dfs(1);

printf("\n");

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

if(reach[i]) count++;

}

if(count==n)

printf("\nGRAPH IS CONNECTED"); else printf("\nGRAPH IS NOT CONNECTED");

return 0;

}

# OUTPUT:

ENTER NUMBER OF VERTICES:4

ENTER THE ADJACENCY MATRIX 11

22

33

44

55

66

77

88

99

12

13

34

56

78

43

21

1->2

2->3

3->4

GRAPH IS CONNECTED

# Binary Search of an ordered array. Iterative and Recursive algorithms are possible. CODE:

#include<stdio.h> #include<conio.h> int main(){

int i, arr[10], search, first, last, middle; printf("Enter 10 elements (in ascending order): "); for(i=0; i<10; i++)

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

printf("\nEnter element to be search: "); scanf("%d", &search);

first = 0, last = 9; middle = (first+last)/2; while(first <= last) {

if(arr[middle]<search) first = middle+1;

else if(arr[middle]==search){

printf("\nThe number, %d found at Position %d", search, middle+1); break; } else

last = middle-1;

middle = (first+last)/2; } if(first>last)

printf("\nThe number, %d is not found in given Array", search); getch();

return 0;}

# OUTPUT:

Enter 10 elements (in ascending order): 1 2

3

4

5

6

7

8

9

10

Enter element to be search: 6

The number, 6 found at Position 6

# Sort a given sequence of numbers using (a) Bubble Sort, and (b) Merge Sort.

**CODE:**

* 1. Bubble sort #include<stdio.h> void bubble(int a[]); int main(){

int i,a[10];

printf("ENTER NUMBERS FOR SORTING : ");

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

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

}

bubble(a);

}

void bubble(int a[]){

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

for(j=0;j<5;j++){

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

temp=a[i]; a[i]=a[j]; a[j]=temp;

}

}

}

printf("\nSORTED ELEMENT ARE\n\n"); for(i=0;i<5;i++){

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

}

}

# OUTPUT:

ENTER A NUMBER: 5 4

3

2

1

SORTED ELEMENT IS 1 2 3 4 5

# CODE:

* 1. Merge sort #include <stdio.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 {

}k++;

}

arr[k] = R[j]; j++;

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);

}

}

void printArray(int A[], int size)

{

int i;

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

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

}

int main()

{

int arr[] = { 55,33,77,22,44};

int arr\_size = sizeof(arr) / sizeof(arr[0]);

printf("GIVEN ARRAY IS\n"); printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

printf("\nSORTED ARRAY IS\n"); printArray(arr, arr\_size);

return 0;

}

# OUTPUT:

GIVEN ARRAY IS 55 33 77 22 44

SORTED ARRAY IS 22 33 44 55 77

# Generate permutations of given n numbers. Iterative and recursive algorithms are possible.

**CODE:**

#include <stdio.h> #include <string.h>

void swap(char \*x, char \*y){ char temp;

temp = \*x;

\*x = \*y;

\*y = temp;

}

void permute(char \*a, int l, int r){ int i;

if (l == r) printf("%s\n", a); else{

for (i = l; i<= r; i++){

swap((a+l), (a+i)); permute(a, l+1, r); swap((a+l), (a+i));

}

}

}

int main(){

char str[] = "ABC"; int n = strlen(str); permute(str, 0, n-1);

return 0;

}

# OUTPUT:

ABC ACB BAC BCA CBA CAB

# Find the closest pair out of given n points in 2-dimensional space.

**CODE:**

#include <stdio.h> #include <float.h> #include <stdlib.h> #include <math.h> struct Point

{

int x, y;

};

int compareX(const void\* a, const void\* b)

{

Point \*p1 = (Point \*)a, \*p2 = (Point \*)b; return (p1->x - p2->x);

}

int compareY(const void\* a, const void\* b)

{

Point \*p1 = (Point \*)a, \*p2 = (Point \*)b; return (p1->y - p2->y);

}

float dist(Point p1, Point p2)

{

return sqrt( (p1.x - p2.x)\*(p1.x - p2.x) +

(p1.y - p2.y)\*(p1.y - p2.y)

);

}

float bruteForce(Point P[], int n)

{

float min = FLT\_MAX; for (int i = 0; i < n; ++i)

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

if (dist(P[i], P[j]) < min)

min = dist(P[i], P[j]);

return min;

}

float min(float x, float y)

{

return (x < y)? x : y;

}

float stripClosest(Point strip[], int size, float d)

{

float min = d;

qsort(strip, size, sizeof(Point), compareY); for (int i = 0; i < size; ++i)

for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j) if (dist(strip[i],strip[j]) < min)

min = dist(strip[i], strip[j]);

return min;

}

float closestUtil(Point P[], int n)

{

if (n <= 3)

return bruteForce(P, n); int mid = n/2;

Point midPoint = P[mid]; float dl = closestUtil(P, mid);

float dr = closestUtil(P + mid, n-mid); float d = min(dl, dr);

Point strip[n]; int j = 0;

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

if (abs(P[i].x - midPoint.x) < d) strip[j] = P[i], j++;

return min(d, stripClosest(strip, j, d) );

}

float closest(Point P[], int n){

qsort(P, n, sizeof(Point), compareX); return closestUtil(P, n);

}

int main(){

Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};

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

printf("THE SMALLEST DISTANCE IS: %f ", closest(P, n)); return 0;

}

# OUTPUT:

THE SMALLEST DISTANCE IS: 1.414214

# Unique partitions of a positive integer.

**CODE:**

#include<stdio.h>

void printarray(int p[], int n){ int i;

for ( i = 0; i< n; i++) printf("%d ", p[i]);

printf("\n");} void partition(int n){ int p[n],k = 0;

p[k] = n;

int rem\_val; while (1){

printarray(p, k+1); rem\_val = 0;

while (k >= 0 && p[k] == 1){ rem\_val += p[k];

k--;}

if (k < 0) return; p[k]--;

rem\_val++;

while (rem\_val> p[k]){

p[k+1] = p[k];

rem\_val = rem\_val - p[k];

k++;}

p[k+1] = rem\_val; k++; } }

int main(){ int num;

printf("\nENTER A NUMBER TO PERFORM INTEGER PARTITION: ");

scanf("%d", &num); partition(num); return 0;}

# OUTPUT:

ENTER A NUMBER TO PERFORM INTEGER PARTITION: 4 4

3 1

2 2

2 1 1

1 1 1 1

# Matrix multiplication using Dynamic Programming algorithm.

**CODE:**

#include <limits.h> #include <stdio.h>

int MatrixChainOrder(int p[], int i, int j)

{

if (i == j)

return 0;

int k;

int min = INT\_MAX; int count;

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

count = MatrixChainOrder(p, i, k) + MatrixChainOrder(p, k + 1, j) +

p[i - 1] \* p[k] \* p[j]; if (count < min)

min = count;

}

return min;

}

int main()

{

int arr[] = { 1, 2, 3, 4, 3 };

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

printf("MINIMUM NUMBER OF MULTIPLICATION IS %d

",MatrixChainOrder(arr, 1, n - 1)); getchar();

return 0;

}

# OUTPUT:

MINIMUM NUMBER OF MULTIPLICATION IS 30

# Solution of Rod-cutting problem using Dynamic Programming algorithm.

**CODE:**

#include<stdio.h> #include<limits.h>

int max(int a, int b) { return (a > b)? a : b;} int cutRod(int price[], int n)

{

int val[n+1]; val[0] = 0; int i, j;

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

{

int max\_val = INT\_MIN; for (j = 0; j < i; j++)

max\_val = max(max\_val, price[j] + val[i-j-1]); val[i] = max\_val;

}

return val[n];

}

int main()

{

int arr[] = {1, 5, 8, 9, 10, 17, 17, 20};

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

printf("MAXIMUM OBTAINABLE VALUE IS %d", cutRod(arr, size)); getchar();

return 0;

}

# OUTPUT:

MAXIMUM OBTAINABLE VALUE IS 22

# Generate pseudo-random numbers.

**CODE:**

#include <stdio.h> #include <stdlib.h> int main()

{

int i;

printf("RANDOM NUMBERS ARE \n\n"); for (i = 0; i<5; i++)

{

printf("%d", rand());

}

return 0;

}

# OUTPUT:

RANDOM NUMBERS ARE 41 18467 6334 26500 19169

# Implement Strassen’s algorithm for matrix multiplication

**CODE:**

#include<stdio.h> int main()

{

int a[2][2], b[2][2], c[2][2], i, j;

int m1, m2, m3, m4 , m5, m6, m7; printf("Enter the 4 elements of first matrix: ");

for(i = 0;i < 2; i++) for(j = 0;j < 2; j++)

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

printf("Enter the 4 elements of second matrix: "); for(i = 0; i< 2; i++)

for(j = 0;j < 2; j++) scanf("%d", &b[i][j]); printf("\nThe first matrix is\n");

for(i = 0; i< 2; i++){

printf("\n");

for(j = 0; j < 2; j++) printf("%d\t", a[i][j]);

}

printf("\nThe second matrix is\n"); for(i = 0;i < 2; i++)

{

printf("\n");

for(j = 0;j < 2; j++) printf("%d\t", b[i][j]);

}

m1= (a[0][0] + a[1][1]) \* (b[0][0] + b[1][1]);

m2= (a[1][0] + a[1][1]) \* b[0][0];

m3= a[0][0] \* (b[0][1] - b[1][1]);

m4= a[1][1] \* (b[1][0] - b[0][0]);

m5= (a[0][0] + a[0][1]) \* b[1][1];

m6= (a[1][0] - a[0][0]) \* (b[0][0]+b[0][1]);

m7= (a[0][1] - a[1][1]) \* (b[1][0]+b[1][1]); c[0][0] = m1 + m4- m5 + m7;

c[0][1] = m3 + m5;

c[1][0] = m2 + m4;

c[1][1] = m1 - m2 + m3 + m6;

printf("\nAfter multiplication using Strassen's algorithm \n"); for(i = 0; i< 2 ; i++)

{

printf("\n");

for(j = 0;j < 2; j++) printf("%d\t", c[i][j]);

}

return 0;

}

# OUTPUT:

Enter the 4 elements of first matrix: 2 3

4

5

Enter the 4 elements of second matrix: 6 7

8

9

The first matrix is 2 3

4 5

The second matrix is 6 7

8 9

After multiplication using Strassen's algorithm 36 41

64 73

# Prim's algorithm to find minimum cost tree (shortest path in a tree).

**CODE:**

#include<stdio.h> #include<conio.h>

int a,b,u,v,n,i,j,ne=1,visited[10]={0},min,mincost=0,cost[10][10]; void main(){

printf("\nENTER THE NUMBER OF NODES:");

scanf("%d",&n);

printf("\nENTER THE ADJACENCY MATRIX\n");

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

scanf("%d",&cost[i][j]); if(cost[i][j]==0)

cost[i][j]=999;}

visited[1]=1; printf("\n"); while(ne < n){

for(i=1,min=999;i<=n;i++) for(j=1;j<=n;j++) if(cost[i][j]< min)

if(visited[i]!=0){

min=cost[i][j]; a=u=i; b=v=j;}

if(visited[u]==0 || visited[v]==0){

printf("\nEDGE %d:(%d %d) COST:%d",ne++,a,b,min); mincost+=min;

visited[b]=1; } cost[a][b]=cost[b][a]=999; }

printf("\n\nMINIMUM COST=%d",mincost); getch(); }

# OUTPUT:

ENTER THE NUMBER OF NODES:2 ENTER THE ADJACENCY MATRIX 3

4

5

6

EDGE 1:(1 2) COST:4 MINIMUM COST=4

# Kruskal's algorithm to find minimum cost tree (shortest path in a tree).

**CODE:**

#include <stdio.h> #include <conio.h> #include <stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9]; int find(int);

int uni(int,int); void main()

{

printf("\n\tIMPLEMENTATION OF KRUSKAL`S ALGORITHM\n");

printf("\nEnter The Number Of Vertices:"); scanf("%d",&n);

printf("\nEnter The Cost Adjacency Matrix\n"); for(i=1;i<=n;i++)

{

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

{

scanf("%d",&cost[i][j]); if(cost[i][j]==0)

cost[i][j]=999;

}

}

printf("The edges of Minimum Cost Spanning Tree are\n"); while(ne < n)

{

for(i=1,min=999;i<=n;i++)

{

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

{

if(cost[i][j] < min)

{

min=cost[i][j]; a=u=i;

b=v=j;

}

}

}

u=find(u); v=find(v); if(uni(u,v))

{

printf("%d Edge (%d,%d) = %d\n",ne++,a,b,min); mincost +=min;

}

cost[a][b]=cost[b][a]=999;

}

printf("\nMinimum Cost = %d\n",mincost); getch();

}

int find(int i)

{

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

}

int uni(int i,int j)

{

if(i!=j)

{

}

parent[j]=i; return 1;

return 0;

}

# OUTPUT:

IMPLEMENTATION OF KRUSKAL`S ALGORITHM

Enter The Number Of Vertices:2 Enter The Cost Adjacency Matrix 22

33

11

44

The edges of Minimum Cost Spanning Tree are 1 Edge (2,1) = 11

Minimum Cost = 11