**Practical 1: Implementation and Time Analysis of sorting Algorithms.**

**#Bubble Sort:-**

**Program:-**

#include<stdio.h>

#include<time.h>

void BubbleSort()

{

int i,n=5,temp,j,arr[5]={10,40,30,50,20};

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

{

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

{

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

{

temp = arr[j];

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

arr[j+1]= temp;

}

}

}

printf("The array sorted in ascending order is: \n");

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

{

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

}

}

int main()

{

clock\_t t;

t = clock();

BubbleSort();

t= clock()- t;

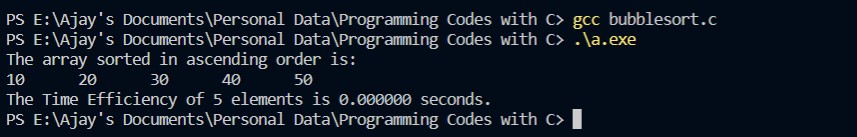
double time\_taken = ((double)t) / CLOCKS\_PER\_SEC; //in sec

printf("\nThe Time Efficiency of 5 elements is %f seconds.\n", time\_taken);

return 0;

}

**Output**:-



**#Selection Sort:-**

**Program:-**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include<time.h>

int smallest(int arr[], int k, int n);

void selection\_sort(int arr[], int n);

void main()

{

clock\_t t;

t = clock();

int arr[5]={10,40,30,50,20}, i, n=5;

selection\_sort(arr, n);

printf("The sorted array is: \n");

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

{

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

}

t= clock()-t;

double time\_taken = ((double)t) / CLOCKS\_PER\_SEC;

printf("\nThe Time Efficiency of 5 elements is %f seconds.\n" ,time\_taken);

}

int smallest(int arr[], int k, int n)

{

int pos = k, small = arr[k], i;

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

{

if (arr[i] < small)

{

small = arr[i];

pos = i;

}

}

return pos;

}

void selection\_sort(int arr[], int n)

{

int k, pos, temp;

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

{

pos = smallest(arr, k, n);

temp = arr[k];

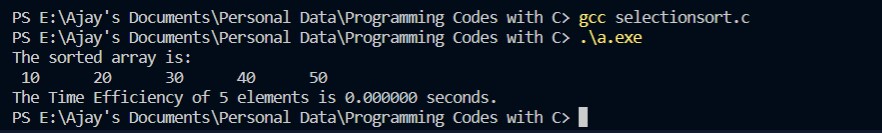
arr[k] = arr[pos];

arr[pos] = temp;

}

}

**Output:-**



**#Selection Sort:-**

**Program:-**

#include<stdio.h>

#include<time.h>

void InsertionSort()

{

int i,n=5,temp,j,arr[5]={10,40,30,50,20};

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

{

temp = arr[i];

j = i-1;

while ((temp< arr[j] )&& (j>=0))

{

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

j--;

}

arr[j+1]= temp;

}

printf("Insertion Sorted array: ");

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

{

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

}

}

int main()

{

clock\_t t;

t = clock();

InsertionSort();

t= clock()-t;

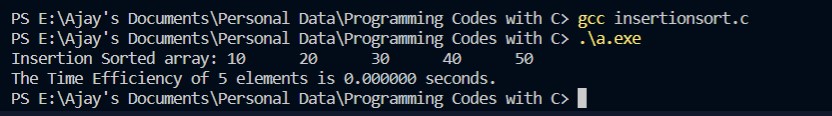
double time\_taken = ((double)t) / CLOCKS\_PER\_SEC; //in sec

printf("\nThe Time Efficiency of 5 elements is %f seconds.\n", time\_taken);

return 0;

}

**Output:-**

****

**#Merge Sort:-**

**Program:-**

#include <stdio.h>

#include <conio.h>

#include<time.h>

#define size 100

void merge(int a[], int, int, int);

void merge\_sort(int a[], int, int);

void main()

{

clock\_t t;

t = clock();

int arr[5]={10,40,50,20,30}, i, n=5;

merge\_sort(arr, 0, n - 1);

printf("Merge sorted array is: \n");

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

{

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

}

t= clock()-t;

double time\_taken = ((double)t) / CLOCKS\_PER\_SEC;

printf("\nThe Time Efficiency of 5 elements is %f seconds.\n" ,time\_taken);

}

void merge(int arr[], int beg, int mid, int end)

{

int i = beg, j = mid + 1, index = beg, temp[size], k;

while ((i <= mid) && (j <= end))

{

if (arr[i] < arr[j])

{

temp[index] = arr[i];

i++;

}

else

{

temp[index] = arr[j];

j++;

}

index++;

}

if (i > mid)

{

while (j <= end)

{

temp[index] = arr[j];

j++;

index++;

}

}

else

{

while (i <= mid)

{

temp[index] = arr[i];

i++;

index++;

}

}

for (k = beg; k < index; k++)

arr[k] = temp[k];

}

void merge\_sort(int arr[], int beg, int end)

{

int mid;

if (beg < end)

{

mid = (beg + end) / 2;

merge\_sort(arr, beg, mid);

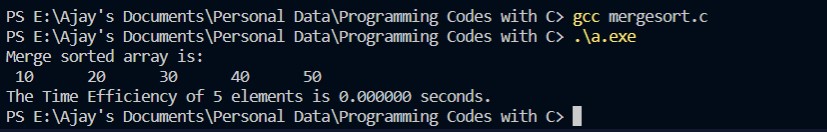
merge\_sort(arr, mid + 1, end);

merge(arr, beg, mid, end);

}

}

**Output:-**

****

**#Quick Sort:-**

**Program:-**

#include <stdio.h>

#include <conio.h>

#include<time.h>

#define size 100

int partition(int a[], int beg, int end);

void quick\_sort(int a[], int beg, int end);

void main()

{

clock\_t t;

t = clock();

int arr[5]={10,30,40,20,50}, i, n=5;

quick\_sort(arr, 0, n - 1);

printf("Quick sorted array is: \n");

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

{

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

}

t= clock()-t;

double time\_taken = ((double)t) / CLOCKS\_PER\_SEC;

printf("\nThe Time Efficiency of 5 elements is %f seconds.\n" ,time\_taken);

}

int partition(int a[], int beg, int end)

{

int left, right, temp, loc, flag;

loc = left = beg;

right = end;

flag = 0;

while (flag != 1)

{

while ((a[loc] <= a[right]) && (loc != right))

right--;

if (loc == right)

flag = 1;

else if (a[loc] > a[right])

{

temp = a[loc];

a[loc] = a[right];

a[right] = temp;

loc = right;

}

if (flag != 1)

{

while ((a[loc] >= a[left]) && (loc != left))

left++;

if (loc == left)

flag = 1;

else if (a[loc] < a[left])

{

temp = a[loc];

a[loc] = a[left];

a[left] = temp;

loc = left;

}

}

}

return loc;

}

void quick\_sort(int a[], int beg, int end)

{

int loc;

if (beg < end)

{

loc = partition(a, beg, end);

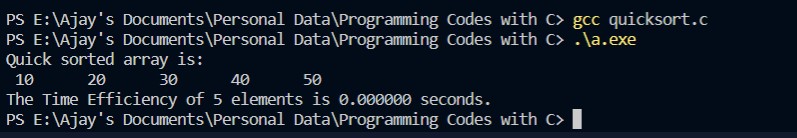
quick\_sort(a, beg, loc - 1);

quick\_sort(a, loc + 1, end);

}

}

**Output:-**

****

**Practical 2: Implementation and Time analysis of linear and binary search algorithm.**

**Program:-**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main()

{

int array[100],search\_key,i,j,n,low,high,location,choice;

void linear\_search(int search\_key, int array[100],int n);

void binary\_search(int search\_key, int array[100],int n);

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

scanf("%d",&n);

printf("ENTER THE ELEMENT OF THE ARRAY: \n");

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

{

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

}

printf("ENTER THE SEARCH KEY: \n");

scanf("%d",&search\_key);

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

printf("1.LINER SEARCH\n");

printf("2.BINARY SEARCH \n");

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

printf("Enter your choice : ");

scanf("%d",&choice);

clock\_t t;

t = clock();

switch (choice)

{

case 1:

linear\_search(search\_key,array,n);

break;

case 2:

binary\_search(search\_key,array,n);

break;

default:

exit(0);

}

t = clock() - t;

double time\_taken = ((double)t) / CLOCKS\_PER\_SEC;

printf("The Time Efficiency of elements is %f seconds.\n",time\_taken);

}

void linear\_search(int search\_key,int array[100],int n)

{

int i,location;

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

{

if(search\_key == array[i])

{

location = i;

}

}

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

printf("YOUR CHOICE IS LINER SEARCHING \n");

printf("THE LOCATION OF SEARCH KEY = %d IS %d \n",search\_key,location);

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

}

void binary\_search(int search\_key,int array[100],int n)

{

int mid,i,low,high;

low=1;

high=n;

mid=(low+high)/2;

while(search\_key != array[mid])

{

if(search\_key <= array[mid])

{

low=1;

high=mid+1;

mid=(low+high)/2;

}

else

{

low=mid+1;

high=n;

mid=(low+high)/2;

}

}

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

printf("YOUR CHOICE IS BINARY SEARCHING \n");

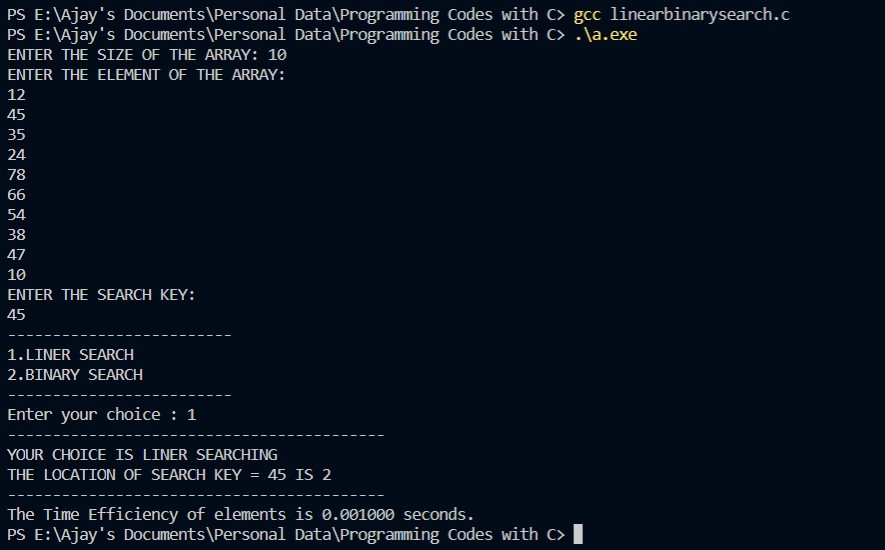
printf("LOCATION : %d \t",mid);

printf("SEARCH KEY : %d found!!!\n",search\_key);

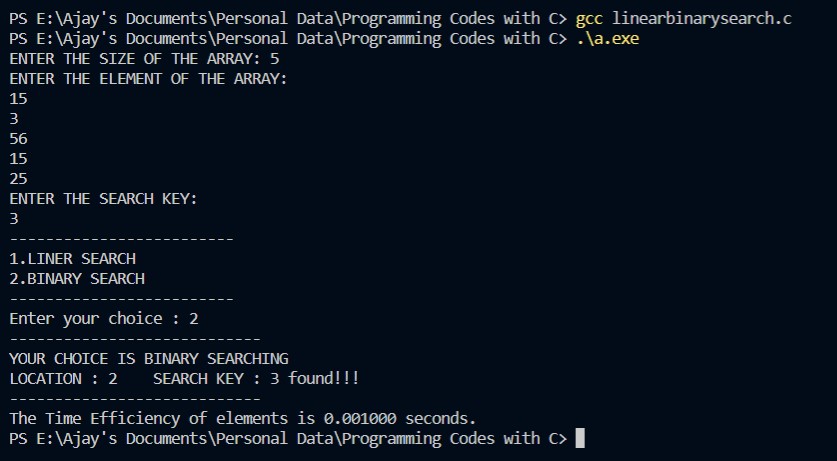
printf("----------------------------\n");

}

**Output 1:-**

****

**Output 2:-**

****

**Practical 3: Implementation of max-heap sort algorithm.**

**Program:-**

#include<stdio.h>

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

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void heapify(int arr[] ,int n,int i)

{

int max = i;

int leftChild = 2\*i+1;

int rightChild = 2\*i+2;

if(leftChild < n && arr[leftChild]>arr[max])

{

max = leftChild;

}

if(rightChild < n && arr[rightChild]>arr[max])

{

max = rightChild;

}

if(max != i)

{

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

heapify(arr,n,max);

}

}

void heapsort(int arr[],int n)

{

for(int i = n/2 -1; i>=0; i--)

{

heapify(arr,n,i);

}

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

{

swap(&arr[0],&arr[i]);

heapify(arr,i,0);

}

}

void display(int arr[] ,int n)

{

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

{

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

}

printf("\n");

}

int main()

{

int arr[]={40,30,50,20,60,10,70};

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

printf("Original Array: \n");

display(arr,n);

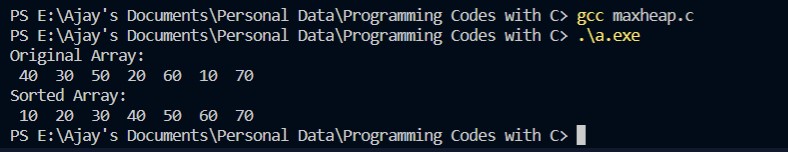
heapsort(arr,n);

printf("Sorted Array: \n");

display(arr,n);

}

**Output:-**

****

**Practical 4: Implementation and Time analysis of factorial program using iterative and recursive method.**

**Program:-**

#include<stdio.h>

#include<conio.h>

void fact(int n)

{

int f = 1;

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

{

f=f\*i;

}

printf("\nThe factorial is %d",f);

}

int rfact(int n1)

{

int factn=1;

if(n1!=0){

factn = rfact(n1-1)\*n1;}

//printf("\nThe factorial is %d",factn);

return factn;

}

int main()

{

int n;

printf("\nEnter the number to find factorial\n");

scanf("\n %d",&n);

printf("\n\nItrative");

fact(n);

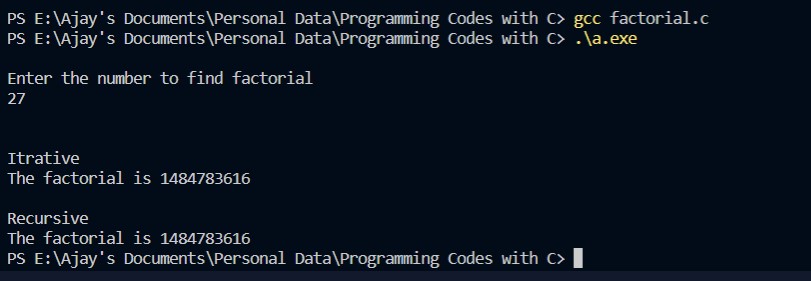
printf("\n\nRecursive \n");

printf("The factorial is %d",rfact(n));

return 0;

}

**Output:-**

****

**Practical 5: Implementation of a knapsack problem using dynamic programming.**

**Program:-**

#include<stdio.h>

#include<conio.h>

int w[10],p[10],v[10][10],n,i,j,cap,x[10]={0};

int max(int i,int j)

{

return ((i>j)?i:j);

}

int knap(int i,int j)

{

int value;

if(v[i][j]<0)

{

if(j<w[i])

{

value=knap(i-1,j);

}

else

{

value=max(knap(i-1,j),p[i]+knap(i-1,j-w[i]));

}

v[i][j]=value;

}

return(v[i][j]);

}

int main()

{

int profit,count=0;

printf("\nEnter the number of elements\n");

scanf("%d",&n);

printf("Enter the profit and weights of the elements\n");

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

{

printf("For item no %d\n",i);

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

}

printf("\nEnter the capacity \n");

scanf("%d",&cap);

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

{

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

{

if((i==0)||(j==0))

{

v[i][j]=0;

}

else

{

v[i][j]=-1;

}

}

}

profit=knap(n,cap);

i=n;

j=cap;

while(j!=0&&i!=0)

{

if(v[i][j]!=v[i-1][j])

{

x[i]=1;

j=j-w[i];

i--;

}

else

{

i--;

}

}

printf("Items included are\n");

printf("Sl.no\tweight\tprofit\n");

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

{

if(x[i])

{

printf("%d\t%d\t%d\n",++count,w[i],p[i]);

}

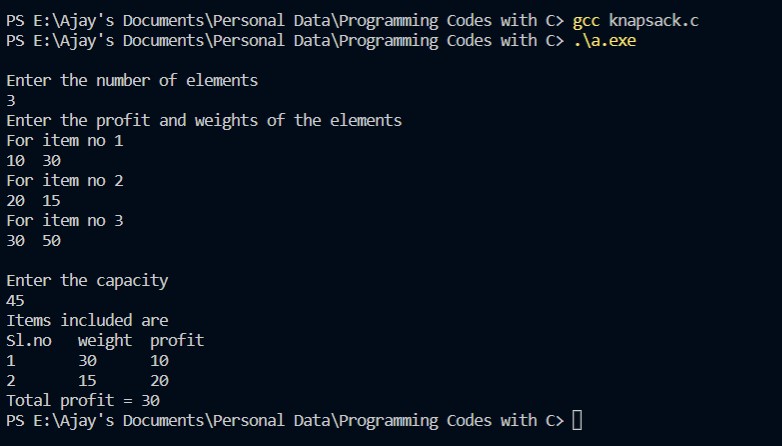
}

printf("Total profit = %d\n",profit);

return 0;

}

**Output:-**

****

**Practical 6: Implementation of chain matrix multiplication using dynamic programming.**

**Program:-**

#include<stdio.h>

#include<limits.h>

int MatrixChainMultiplication(int p[], int n)

{

int m[n][n];

int i, j, k, L, q;

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

{

m[i][i] = 0;

}

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

{

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

{

j = i+L-1;

m[i][j] = INT\_MAX;

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

{

q = m[i][k] + m[k+1][j] + p[i-1]\*p[k]\*p[j];

if (q < m[i][j])

{

m[i][j] = q;

}

}

}

}

return m[1][n-1];

}

int main()

{

int n,i;

printf("Enter number of matrices\n");

scanf("%d",&n);

n++;

int arr[n];

printf("Enter dimensions \n");

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

{

printf("Enter d%d :: ",i);

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

}

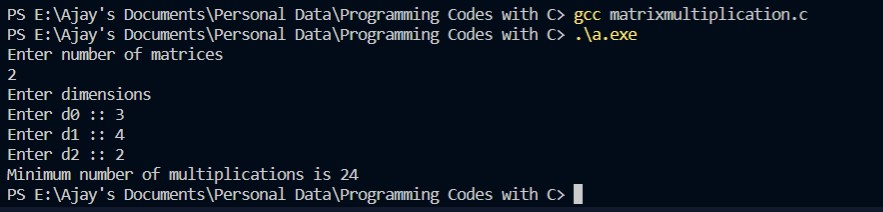
int size = sizeof(arr)/sizeof(arr[0]);

printf("Minimum number of multiplications is %d ", MatrixChainMultiplication(arr, size));

return 0;

}

**Output:-**

****

**Practical 7: Implementation of making a change problem using dynamic programming.**

**Program:-**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int m,n,i,j;

printf("Enter the amount of which you want change\n");

scanf("%d",&m);

printf("\nEnter the number of determinants\n");

scanf("%d",&n);

int c[n+1][m+1],dc[n+1];

printf("\nEnter the determinants\n");

dc[0]=0;

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

{

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

}

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

{

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

{

c[i][j]=0;

}

}

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

{

c[i][0]=0;

}

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

{

c[0][i]=i;

}

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

{

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

{

if(dc[1]>c[0][j])

{

c[i][j]=0;

}

else if(dc[i]<=j)

{

if((c[i][j-dc[i]] +1 )<c[i-1][j])

{

c[i][j]=( c[i][j-dc[i]] +1 );

}

}

else

{

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

}

}

}

printf("\n");

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

{

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

for(j=0;j<m+1;j++)

{

printf(" %d",c[i][j]);

}

printf("\n");

}

printf("\nThe change you get is \n");

i=n;

j=m;

while(i!=1)

{

if(c[i][j]==c[i-1][j])

{

i=i-1;

}

else

{

j=j-dc[i];

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

}

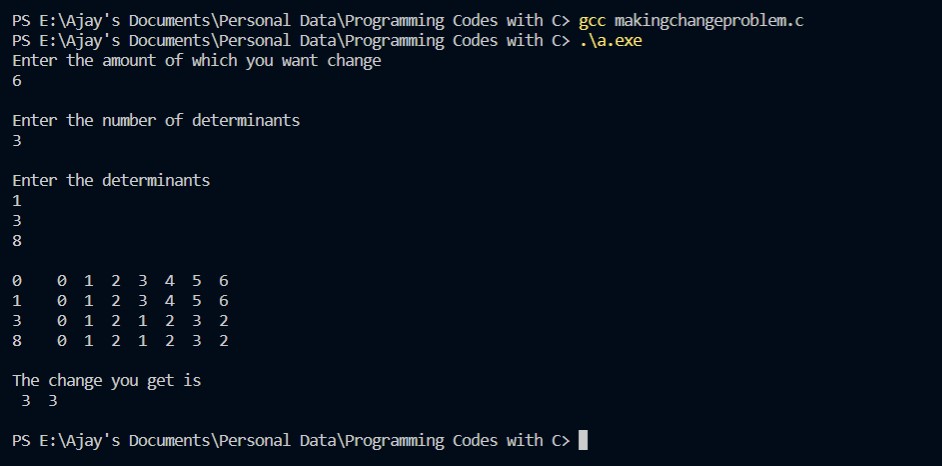
}

printf("\n\n");

return 0;

}

**Output:-**

****

**Practical 8: Implementation of a knapsack problem using greedy algorithm.**

**Program:-**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int m,n,i,j;

printf("Enter maximum weight of knapsack ");

scanf("%d",&m);

printf("\nEnter number of objects ");

scanf("%d",&n);

int wt=0,k=0;

float cal[n],p[n],w[n],x[n],prof=0;

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

{

x[i]=0;

}

printf("\nEnter weights\n");

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

{

printf("w[%d] = ",i);

scanf("%f",&w[i]);

}

printf("\nEnter profits\n");

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

{

printf("p[%d] = ",i);

scanf("%f",&p[i]);

}

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

{

cal[i]=p[i]/w[i];

}

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

{

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

{

if(cal[i]<cal[j])

{

int t1,t2,t3;

t1=cal[i]; cal[i]=cal[j]; cal[j]=t1;

t2=w[i]; w[i]=w[j]; w[j]=t2;

t3=p[i]; p[i]=p[j]; p[j]=t3;

}

}

}

printf("\n\n p[i]\t\t w[i]\t\t cal[i]\n");

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

{

printf("%f\t %f\t %f\t\n",p[i],w[i],cal[i]);

}

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

{

if((wt+w[i])<=m)

{

k++; x[i]=1; wt+=w[i]; prof+=p[i];

}

else

{

k++; x[i]=(m-wt)/w[i]; w[i]=m-wt; wt=m; prof+=(x[i]\*p[i]); p[i]=(x[i]\*p[i]); break;

}

}

printf("\nThe selected weights are \n\ni\t w[i]\t\t p[i]\n");

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

{

printf("%d\t%f\t%f\n",i+1,w[i],p[i]);

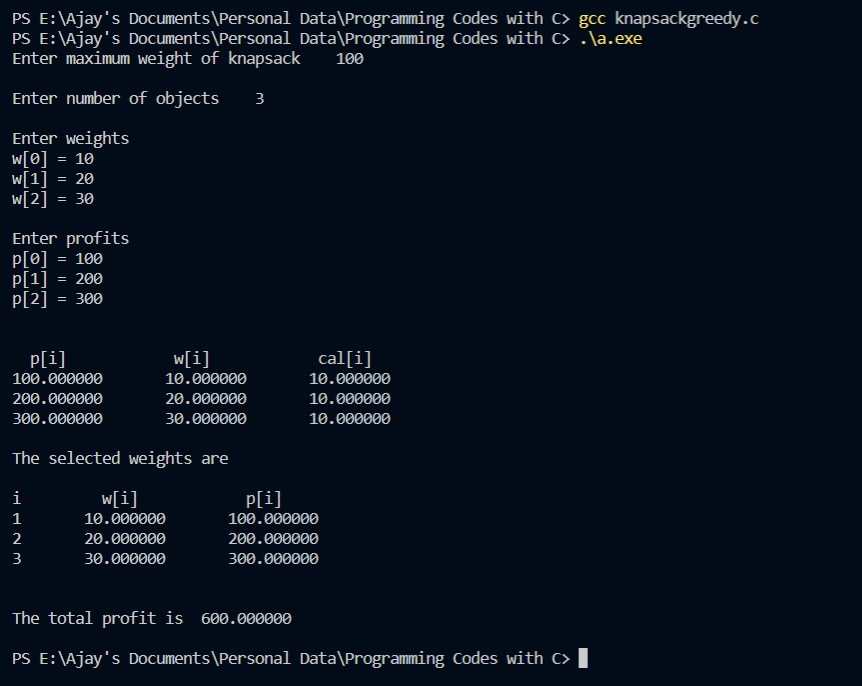
}

printf("\n\nThe total profit is %f\n\n",prof);

return 0;

}

**Output:-**

****

**Practical 9: Implementation of Graph and Searching (DFS and BFS).**

**#For DFS:**

**Program:-**

#include<stdio.h>

#include<conio.h>

int a[20][20],reach[20],n;

int 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("\n Enter 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("\n Enter 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("\n Graph is connected");

}

else

{

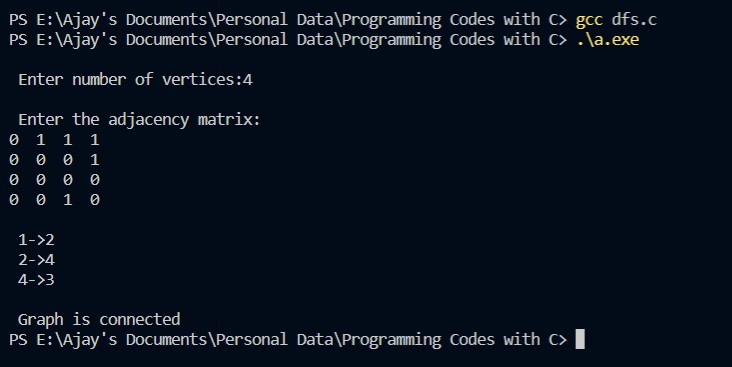
printf("\n Graph is not connected");

}

return 0;

}

**Output:-**

****

**#For BFS:**

**Program:-**

#include<stdio.h>

#include<conio.h>

int a[20][20],visited[20],n,i,j,f=0,r=-1,q[20];

int bfs(int v)

{

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

{

if(a[v][i] && !visited[i])

{

q[++r]=i;

if(f<=r)

{

visited[q[f]]=1;

bfs(q[f++]);

}

}

}

}

int main()

{

int v;

printf("\nEnter number of vertices: ");

scanf("%d",&n);

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

{

q[i]=0;

visited[i]=0;

}

printf("\nEnter adjacency matrix: \n");

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

{

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

{

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

}

}

printf("Enter starting vertex: ");

scanf("%d",&v);

bfs(v);

printf("The node which are reachable are: ");

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

{

if(visited[i])

{

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

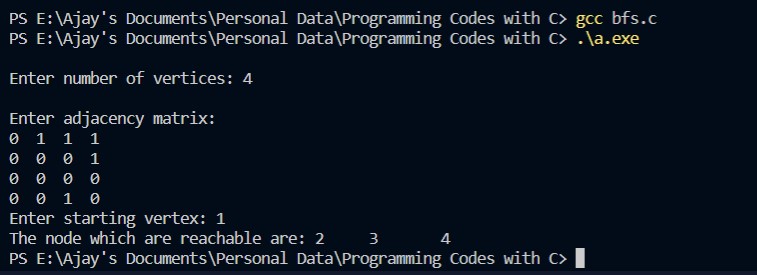
}

}

return 0;

}

**Output:-**

****

**Practical 10: Implement prim’s algorithm.**

**Program:-**

#include<stdio.h>

#include<stdlib.h>

#define infinity 9999

#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();

int main()

{

int i,j,total\_cost;

printf("Enter no. of vertices:");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

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

{

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

{

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

}

}

total\_cost=prims();

printf("\nspanning tree matrix:\n");

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

{

printf("\n");

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

{

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

}

}

printf("\n\nTotal cost of spanning tree=%d",total\_cost);

return 0;

}

int prims()

{

int cost[MAX][MAX];

int u,v,min\_distance,distance[MAX],from[MAX];

int visited[MAX],no\_of\_edges,i,min\_cost,j;

//create cost[][] matrix,spanning[][]

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

{

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

{

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

{

cost[i][j]=infinity;

}

else

{

cost[i][j]=G[i][j];

spanning[i][j]=0;

}

}

}

distance[0]=0;

visited[0]=1;

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

{

distance[i]=cost[0][i];

from[i]=0;

visited[i]=0;

}

min\_cost=0;

no\_of\_edges=n-1;

while(no\_of\_edges>0)

{

min\_distance=infinity;

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

{

if(visited[i]==0&&distance[i]<min\_distance)

{

v=i;

min\_distance=distance[i];

}

}

u=from[v];

spanning[u][v]=distance[v];

spanning[v][u]=distance[v];

no\_of\_edges--;

visited[v]=1;

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

{

if(visited[i]==0&&cost[i][v]<distance[i])

{

distance[i]=cost[i][v];

from[i]=v;

}

}

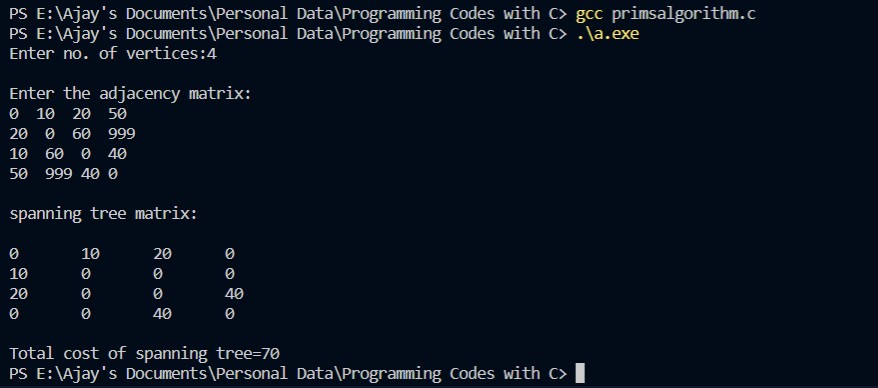
min\_cost=min\_cost+cost[u][v];

}

return(min\_cost);

}

**Output:-**

****

**Practical 11: Implement kruskal’s algorithm.**

**Program:-**

#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);

int main()

{

printf("\nImplementation of Kruskal's algorithm: \n\n");

printf("\nEnter the no. 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("\nThe edges of Minimum Cost Spanning Tree are: \n\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("\n%d edge (%d,%d) =%d\n",ne++,a,b,min);

mincost +=min;

}

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

}

printf("\n\tMinimum cost = %d\n",mincost);

return 0;

}

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:-**

****

**Practical 12: Implement LCS problem.**

**Program:-**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

int max(int a, int b);

int lcs( char \*X, char \*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 max(int a, int b)

{

return (a > b)? a : b;

}

int main()

{

char X[] = "AGGTAB";

char Y[] = "GXTXAYB";

int m = strlen(X);

int n = strlen(Y);

printf("Length of LCS is %d", lcs( X, Y, m, n ) );

return 0;

}

Output:-

