1.FIND THE SUM OF N NUMBERS USING ARRAYS, FIND THE TIME COMPLEXITY

```
#include <stdio.h>
int main()
{
int n, sum = 0, c, value;
printf("How many numbers you want to add?\n");
scanf("%d", &n); printf("Enter %d integers\n", n);
for (c = 1; c <= n; c++)
{
scanf("%d", &value);
sum = sum + value;
}
printf("Sum of the integers = %d\n", sum);
return 0;
}
```

2.WRITE A PROGRAM THAT INPUTS THREE INTEGERS AND OUTPUTS THEM IN NON DECREASING ORDER. FIND THE TIME COMPLEXITY.

```
#include<stdio.h>
int main()
{
int a[3],i,j,item;
printf("Enter the 3 integers: ");
scanf("%d %d %d",&a[1],&a[2],&a[3]);
for(j=2;j<=3;j++)
{
item=a[j];
i=j-1;
while(i>=1 && item<a[i])
{
a[i+1]=a[i];
i=i-1;
}
a[i+1]=item;
}
printf("NonDecreasing order of integers is %d,%d,%d\n",a[1],a[2],a[3]);
}
```

3.FIND THE FACTORIAL OF A NUMBER USING TWO METHODS, RECURSIVE AND ITERATIVE. FIND THE TIME COMPLEXITY

I)RECURSIVE

```
#include<stdio.h>
long int multiplyNumbers(int n);
```

```
int main()
{
int n;
printf("Enter a positive integer: ");
scanf("%d",&n);
printf("Factorial of %d = %ld", n, multiplyNumbers(n));
return 0;
}
long int multiplyNumbers(int n) {
if (n>=1)
return n*multiplyNumbers(n-1);
else
return 1;
}
```

II)ITERATIVE

```
#include <stdio.h>
int main() {
int n, i;
unsigned long long fact = 1;
printf("Enter an integer: ");
scanf("%d", &n);
f (n < 0)
printf("Error! Factorial of a negative number doesn't exist.");
else {
for (i = 1; i <= n; ++i) {
fact *= i;
}
printf("Factorial of %d = %llu", n, fact);
}
return 0;
}
```

4.FIND THE FIBONACCI NUMBERS UP TO N. FIND THE TIME COMPLEXITY.

```
#include <stdio.h>
int main() {
int i, n, t1 = 0, t2 = 1, nextTerm;
printf("Enter the number of terms: ");
scanf("%d", &n);
printf("Fibonacci Series: ");
for (i = 1; i <= n; ++i) {
printf("%d, ", t1);
nextTerm = t1 + t2;
t1 = t2;
t2 = nextTerm;
}
return 0;
}
```

5.FIND THE SUM, PRODUCT OF TWO MATRICES USING RECURSIVE AND ITERATIVE METHODS. FIND TIME COMPLEXITY.

a) PRODUCT:

I) RECURSIVE:

```
#include<stdio.h>
#define MAX 10
void matrixMultiply(int[MAX][MAX], int[MAX][MAX]);
int row1, row2, col1, col2;
int res[MAX][MAX];
int main() {
int mat1[MAX][MAX], mat2[MAX][MAX], i, j, k;
printf("Enter the row and column of first matrix: ");
scanf("%d%d", &row1, &col1);
printf("Enter the row and column of second matrix: ");
scanf("%d%d", &row2, &col2);
if (col1 != row2) {
printf("Matrix multiplication is not possible");
} else {
printf("Enter the First matrix: ");
for (i = 0; i < row1; i++) {
for (j = 0; j < col1; j++) {
scanf("%d", &mat1[i][j]);
}
}
printf("Enter the Second matrix: ");
for (i = 0; i < row2; i++) {
for (j = 0; j < col2; j++) {
scanf("%d", &mat2[i][j]);
}
}
printf("\nThe First matrix is: n");
for (i = 0; i < row1; i++) {
printf("\n");
for (j = 0; j < col1; j++) {
printf("%d ", mat1[i][j]);
}
}
printf("\nThe Second matrix is: n");
for (i = 0; i < row2; i++) {
printf("\n");
for (j = 0; j < col2; j++) {
printf("%d ", mat2[i][j]);
}
}
matrixMultiply(mat1, mat2);
}
printf("\nThe multiplication of two matrixes is : \n");
for (i = 0; i < row1; i++) {
printf("\n");
for (j = 0; j < col2; j++) {
printf("%d ", res[i][j]);
}
}
return 0;
}
void matrixMultiply(int a[MAX][MAX], int b[MAX][MAX]) {
static int sum, i = 0, j = 0, k = 0;
if (i < row1) {
if (j < col2) {
if (k < col1) {
sum = sum + a[i][k] * b[k][j];
k++;
matrixMultiply(a, b);
}
res[i][j] = sum;
sum = 0;
k = 0;
j++;
matrixMultiply(a, b);
}
j = 0;
i++;
matrixMultiply(a, b);
}
}
```

II)ITERATIVE:

```
#include <stdio.h>
```

```
int main()
{
int m, n, p, q, c, d, k, sum = 0;
int first[10][10], second[10][10], multiply[10][10];
printf("Enter the number of rows and columns of first matrix\n");
scanf("%d%d", &m, &n);
printf("Enter the elements of first matrix\n");
for ( c = 0 ; c < m ; c++ )
for ( d = 0 ; d < n ; d++ )
scanf("%d", &first[c][d]);
printf("Enter the number of rows and columns of second matrix\n");
scanf("%d%d", &p, &q);
if ( n != p )
printf("Matrices with entered orders can't be multiplied with each other.\n");
else
{
printf("Enter the elements of second matrix\n");
for ( c = 0 ; c < p ; c++ )
for ( d = 0 ; d < q ; d++ )
scanf("%d", &second[c][d]);
for ( c = 0 ; c < m ; c++ )
{
for ( d = 0 ; d < q ; d++ )
{
for ( k = 0 ; k < p ; k++ )
{
sum = sum + first[c][k]*second[k][d];
}
multiply[c][d] = sum;
sum = 0;
}
}
printf("Product of entered matrices:-\n");
for ( c = 0 ; c < m ; c++ )
{
for ( d = 0 ; d < q ; d++ )
printf("%d\t", multiply[c][d]);
printf("\n");
}
}
return 0;
}
```

B) SUM

I) RECURSIVE

```
#include <stdio.h>
void add( int [][10], int, int, int [][10], int [][10]);
void display(int, int, int[][10]);
int main()
{
int a[10][10], b[10][10], c[10][10] = {0};
int m, n, i, j;
printf("Enter rows and columns for Matrices: ");
scanf("%d%d", &m, &n);
printf("Enter elements in Matrix A:\n");
for (i = 0; i < m; i++)
for (j = 0; j < n; j++)
{
scanf("%d", &a[i][j]);
}
printf("\nEnter elements in Matrix B:\n");
for (i = 0; i < m; i++)
for (j = 0; j < n; j++)
{
scanf("%d", &b[i][j]);
}
add( a, m, n, b, c);
printf("On matrix addition of A and B the result is:\n");
display(m, n, c);
}
void add(int a[10][10], int m, int n, int b[10][10], int c[10][10])
{
static int i = 0, j = 0;
if (i < m)
{
if (j < n)
{
c[i][j] += a[i][j] + b[i][j];
j++;
add(a, m, n, b, c);
}
j = 0;
i++;
add( a, m, n, b, c);
}
}
void display(int m, int n, int c[10][10])
{
int i, j;
for (i = 0; i < m; i++)
{
for (j = 0; j < n; j++)
{
printf("%d ", c[i][j]);
}
printf("\n");
}
}
```

II) ITERATIVE

```
#include <stdio.h>
int main()
{
int m, n, c, d, first[10][10], second[10][10], sum[10][10];
printf("Enter the number of rows and columns of matrix\n");
scanf("%d%d", &m, &n);
printf("Enter the elements of first matrix\n");
for (c = 0; c < m; c++)
for (d = 0; d < n; d++)
scanf("%d", &first[c][d]);
printf("Enter the elements of second matrix\n");
for (c = 0; c < m; c++)
for (d = 0 ; d < n; d++)
scanf("%d", &second[c][d]);
printf("Sum of entered matrices:-\n");
for (c = 0; c < m; c++) {
for (d = 0 ; d < n; d++) {
sum[c][d] = first[c][d] + second[c][d];
printf("%d\t", sum[c][d]);
}
printf("\n");
}
return 0;
}
```

DIVIDE & CONQUER

6.FIND THE MINIMUM AND MAXIMUM NUMBERS IN THE GIVEN ARRAY.

```
#include<stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j)
{
int max1, min1, mid;
if(i==j)
{
max = min = a[i];
}
else
{
if(i == j-1)
{
if(a[i] <a[j])
{
max = a[j];
min = a[i];
}
else
{
max = a[i];
min = a[j];
}
}
else
{
mid = (i+j)/2;
maxmin(i, mid);
max1 = max; min1 = min;
maxmin(mid+1, j);
if(max <max1)
max = max1;
if(min > min1)
min = min1;
}
}
}
int main ()
{
```

```
int i, num;
printf ("\nEnter the total number of numbers : ");
scanf ("%d",&num);
printf ("Enter the numbers : \n");
for (i=1;i<=num;i++)
scanf ("%d",&a[i]);
max = a[0];
min = a[0];
maxmin(1, num);
printf ("Minimum element in an array : %d\n", min);
printf ("Maximum element in an array : %d\n", max);
return 0;
}
```

7.FIND THE POSITION OF THE ELEMENT USING BINARY SEARCH

```
#include <stdio.h>
int main()
{
int c, first, last, middle, n, search, array[100];
printf("Enter number of elements\n");
scanf("%d", &n);
printf("Enter %d integers\n", n);
for (c = 0; c < n; c++)
scanf("%d", &array[c]);
printf("Enter value to find\n");
scanf("%d", &search);
first = 0;
last = n - 1;
middle = (first+last)/2;
while (first <= last) {
if (array[middle] < search)
first = middle + 1;
else if (array[middle] == search) {
printf("%d found at location %d.\n", search, middle+1);
break;
}
else
last = middle - 1;
middle = (first + last)/2;
}
if (first > last)
printf("Not found! %d isn't present in the list.\n", search);
return 0;
}
```
8.SORT THE LIST OF ELEMENTS USING MERGE SORT
```
#include<stdio.h>
void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);
int main()
{
int a[30],n,i;
printf("Enter no of elements:");
scanf("%d",&n);
printf("Enter array elements:");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
mergesort(a,0,n-1);
printf("\nSorted array is :");
for(i=0;i<n;i++)
printf("%d ",a[i]);
return 0;
}
void mergesort(int a[],int i,int j)
{
int mid;
if(i<j)
{
mid=(i+j)/2;
mergesort(a,i,mid);
mergesort(a,mid+1,j);
merge(a,i,mid,mid+1,j);
}
}
void merge(int a[],int i1,int j1,int i2,int j2)
{
int temp[50];
int i,j,k;
i=i1;
j=i2;
k=0;
while(i<=j1 && j<=j2)
{
if(a[i]<a[j])
temp[k++]=a[i++];
else
temp[k++]=a[j++];
}
while(i<=j1)
temp[k++]=a[i++];
while(j<=j2)
temp[k++]=a[j++];
for(i=i1,j=0;i<=j2;i++,j++)
a[i]=temp[j];
}
```

9.SORT THE LIST OF ELEMENTS USING QUICK SORT

```
#include<stdio.h>
void quicksort(int number[25],int first,int last){
int i, j, pivot, temp;
if(first<last){
pivot=first;
i=first;
j=last;
while(i<j){
while(number[i]<=number[pivot]&&i<last)
i++;
while(number[j]>number[pivot])
j--;
if(i<j){
temp=number[i];
number[i]=number[j];
number[j]=temp;
}
}
temp=number[pivot];
number[pivot]=number[j];
number[j]=temp;
quicksort(number,first,j-1);
quicksort(number,j+1,last);
}
}
int main(){
int i, count, number[25];
printf("no.of elements: ");
scanf("%d",&count);
printf("Enter %d elements: ", count);
for(i=0;i<count;i++)
scanf("%d",&number[i]);
quicksort(number,0,count-1);
printf("Order of Sorted elements: ");
for(i=0;i<count;i++)
printf(" %d",number[i]);
return 0;
}
```

10.SORT THE LIST OF ELEMENTS USING SELECTION SORT.

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 7
int intArray[MAX] = {4,6,3,2,1,9,7};
void printline(int count) {
int i;
for(i = 0;i < count-1;i++) {
printf("=");
}
printf("=\n");
}
void display() {
int i;
printf("[");
for(i = 0;i < MAX;i++) {
printf("%d ", intArray[i]);
}
printf("]\n");
}
void selectionSort() {
int indexMin,i,j;
for(i = 0; i < MAX-1; i++) {
indexMin = i;
for(j = i+1;j < MAX;j++) {
if(intArray[j] < intArray[indexMin]) {
indexMin = j;
}
}
if(indexMin != i) {
printf("Items swapped: [ %d, %d ]\n" , intArray[i], intArray[indexMin]);
int temp = intArray[indexMin];
intArray[indexMin] = intArray[i];
intArray[i] = temp;
}
printf("Iteration %d#:",(i+1));
display();
}
}
void main() {
printf("Input Array: ");
display();
printline(50);
selectionSort();
printf("Output Array: ");
display();
printline(50);
}
```

11.SORT THE LIST OF ELEMENTS USING INSERTION SORT.

```
#include<stdio.h>
int main(){
int i, j, count, temp, number[25];
printf("How many numbers u are going to enter?: ");
scanf("%d",&count);
printf("Enter %d elements: ", count);
for(i=0;i<count;i++)
scanf("%d",&number[i]);
for(i=1;i<count;i++){
temp=number[i];
j=i-1;
while((temp<number[j])&&(j>=0)){
number[j+1]=number[j];
j=j-1;
}
number[j+1]=temp;
}
printf("Order of Sorted elements: ");
for(i=0;i<count;i++)
printf(" %d",number[i]);
return 0;
}
```

12.FIND THE PRODUCT OF TWO 2× 2 MATRICES USING STRASSEN MATRIX MULTIPLICATION.

```
#include <stdio.h>
#include <stdlib.h>
void Strassen(int mx1[][2],int mx2[][2],int n)
{
int a,b,c,d,e,f,g,h,P1,P2,P3,P4,P5,P6,P7,Q1,Q2,Q3,Q4,R[2][2],i,j;
a = mx1[0][0] ;
b = mx1[0][1] ;
c = mx1[1][0] ;
d = mx1[1][1] ;
e = mx2[0][0] ;
f = mx2[0][1] ;
g = mx2[1][0] ;
h = mx2[1][1] ;
P1 = a *( f - h ) ;
P2 = ( a + b ) * h ;
P3 = ( c + d ) * e ;
P4 = d * ( g - e ) ;
P5 = ( a + d ) * ( e + h ) ;
P6 = ( b - d ) * ( g + h ) ;
P7 = ( a - c ) * ( e + f ) ;
Q1 = P5 + P4 + P6 - P2 ;
Q2 = P1 + P2 ;
Q3 = P3 + P4 ;
Q4 = P1 + P5 - P3 - P7 ;
R[0][0] = Q1; R[0][1] = Q2; R[1][0] = Q3; R[1][1] = Q4;
printf("\n");
for( i = 0 ; i < n ; i++ )
for( j = 0 ; j < n ; j++ )
printf(" %d ",R[i][j]);
}
int main()
{
int n,MX1[2][2],MX2[2][2],i,j;
printf("\n\t\tSTRASSEN'S ALGORITHM\n\nEnter the size of the matrix: ");
scanf("%d",&n);
printf("Enter the elements of the first matrix: ");
for( i = 0 ; i < n ; i++ )
for( j = 0 ; j < n ; j++ )
scanf("%d",&MX1[i][j]);
printf("Enter the elements of the second matrix");
for( i = 0 ; i < n ; i++ )
for( j = 0 ; j < n ; j++ )
scanf("%d",&MX2[i][j]);
printf("\nThe resultant matrix is :");
Strassen(MX1,MX2,n);
return 0;
}
```

GREEDY METHOD

13. FIND THE SOLUTION OF FRACTIONAL KNAPSACK PROBLEM

```
# include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity) {
float x[20], tp = 0;
int i, j, u;
u = capacity;
for (i = 0; i < n; i++)
x[i] = 0.0;
for (i = 0; i < n; i++) {
if (weight[i] > u)
break;
else {
x[i] = 1.0;
tp = tp + profit[i];
u = u - weight[i];
}
}
if (i < n)
x[i] = u / weight[i];
tp = tp + (x[i] * profit[i]);
printf("\nThe result vector is:- ");
for (i = 0; i < n; i++)
printf("%f\t", x[i]);
printf("\nMaximum profit is:- %f", tp);
}
int main() {
float weight[20], profit[20], capacity;
int num, i, j;
float ratio[20], temp;
printf("\nEnter the no. of objects:- ");
scanf("%d", &num);
printf("\nEnter the wts and profits of each object:- ");
for (i = 0; i < num; i++) {
scanf("%f %f", &weight[i], &profit[i]);
}
printf("\nEnter the capacityacity of knapsack:- ");
scanf("%f", &capacity);
for (i = 0; i < num; i++) {
ratio[i] = profit[i] / weight[i];
}
for (i = 0; i < num; i++) {
for (j = i + 1; j < num; j++) {
if (ratio[i] < ratio[j]) {
temp = ratio[j];
ratio[j] = ratio[i];
ratio[i] = temp;
temp = weight[j];
weight[j] = weight[i];
weight[i] = temp;
temp = profit[j];
profit[j] = profit[i];
profit[i] = temp;
}
}
}
knapsack(num, weight, profit, capacity);
return(0);
}
```

14.FIND THE SOLUTION TO JOB SEQUENCING WITH DEADLINES.

```
#include<stdio.h>
int jobsequence(int d[6],int j[6],int n)
{
int q,i,r,k;
d[0]=0;
j[0]=0;
j[1]=1;
k=1;
for(i=2;i<=n;i++)
{
r=k;
while((d[j[r]]>d[i]) &&(d[j[r]]!=r))
r=r-1;
if((d[j[r]]<=d[i]) && (d[i]>r))
{
for(q=k;q>=r+1;q--)
{
j[q+1]=j[q];
}
j[r+1]=i;
k=k+1;
}
}
return k;
}
void main( )
{
int d[6],j[6],p[6],k,i; printf("Enter the deadlines :");
for(i=1;i<=5;i++)
scanf("%d",&d[i]); printf("Enter the profits :");
for(i=1;i<=5;i++)
scanf("%d",&p[i]);
for(i=1;i<=5;i++)
j[i]=i;
k=jobsequence(d,j,5);
printf("\nThe solution job sequence is ");
for(i=1;i<=k;i++)
printf("\n%d",j[i]);
}
```

15.SORT THE LIST OF ELEMENTS USING HEAP SORT.

```
#include<stdio.h>
void create(int []);
void down_adjust(int [],int);
int main()
{
int heap[30],n,i,last,temp;
printf("Enter no. of elements:");
scanf("%d",&n);
printf("\nEnter elements:");
for(i=1;i<=n;i++)
scanf("%d",&heap[i]);
heap[0]=n;
create(heap);
while(heap[0] > 1)
{
last=heap[0];
temp=heap[1];
heap[1]=heap[last];
heap[last]=temp;
heap[0]--;
down_adjust(heap,1);
}
printf("\nArray after sorting:\n");
for(i=1;i<=n;i++)
printf("%d ",heap[i]);
return 0;
}
void create(int heap[])
{
int i,n;
n=heap[0];
for(i=n/2;i>=1;i--)
down_adjust(heap,i);
}
void down_adjust(int heap[],int i)
{
int j,temp,n,flag=1;
n=heap[0];
while(2*i<=n && flag==1)
{
j=2*i;
if(j+1<=n && heap[j+1] > heap[j])
j=j+1;
if(heap[i] > heap[j])
flag=0;
else
{
temp=heap[i];
heap[i]=heap[j];
heap[j]=temp;
i=j;
}
}
}
```

16.CONSTRUCT THE MINIMUM SPANNING TREE USING PRIM’S ALGORITHM

```
#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;
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);
}
```

17.CONSTRUCT THE MINIMUM SPANNING TREE USING KRUSKAL'S ALGORITHM

```
#include<stdio.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 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("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("\n\tMinimum cost = %d\n",mincost);
}
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;
}
```

18.SOLVE THE SINGLE SOURCE SHORTEST PATH PROBLEM (DIJKSTRA'S ALGORITHM)

```
#include<stdio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);
int main()
{
int G[MAX][MAX],i,j,n,u;
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]);
printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);
return 0;
}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{
int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
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];
for(i=0;i<n;i++)
{
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
{
mindistance=INFINITY;
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
j=pred[j];
printf("<-%d",j);
}while(j!=startnode);
}
}
```

DYNAMIC PROGRAMMING

19.FIND THE OPTIMAL SEQUENCE OF MATRIX MULTIPLICATION USING MATRIX CHAIN MULTIPLICATION METHOD.

```
#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);
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;
}
```

20.CONSTRUCT THE OPTIMAL BINARY SEARCH TREE FOR THE GIVEN INPUT.

```
#include<stdio.h>
#define MAX 10
void main()
{
char ele[MAX][MAX];
int w[MAX][MAX], c[MAX][MAX], r[MAX][MAX], p[MAX], q[MAX];
int temp=0, root, min, min1, n;
int i,j,k,b;
printf("Enter the number of elements:");
scanf("%d",&n);
printf("\n");
for(i=1; i <= n; i++)
{
printf("Enter the Element of %d:",i);
scanf("%d",&p[i]);
}
printf("\n");
for(i=0; i <= n; i++)
{
printf("Enter the Probability of %d:",i);
scanf("%d",&q[i]);
}
printf("W\t\tC\t\tR\n");
for(i=0; i <= n; i++)
{
for(j=0; j <= n; j++)
{
if(i == j)
{
```

```
w[i][j] = q[i];
c[i][j] = 0;
r[i][j] = 0;
printf("W[%d][%d]: %d\tC[%d][%d]: %d\tR[%d][%d]: %d\n",i,j,w[i][j],i,j,c[i][j],i,j,r[i][j]);
}
}
}
printf("\n");
for(b=0; b < n; b++)
{
for(i=0,j=b+1; j < n+1 && i < n+1; j++,i++)
{
if(i!=j && i < j)
{
w[i][j] = p[j] + q[j] + w[i][j-1];
min = 30000;
for(k = i+1; k <= j; k++)
{
min1 = c[i][k-1] + c[k][j] + w[i][j];
if(min > min1)
{
min = min1;
temp = k;
}
}
c[i][j] = min;
r[i][j] = temp;
}
printf("W[%d][%d]: %d\tC[%d][%d]: %d\tR[%d][%d]: %d\n",i,j,w[i][j],i,j,c[i][j],i,j,r[i][j]);
}
printf("\n");
}
printf("Minimum cost = %d\n",c[0][n]);
root = r[0][n];
printf("Root = %d \n",root);
}
```

21.FIND THE SOLUTION OF 0/1 KNAPSACK PROBLEM.

```
#include<stdio.h>
int max(int a, int b) { return (a > b)? a : b; }
int knapSack(int W, int wt[], int val[], int n)
{
```

```
int i, w;
int K[n+1][W+1];
for (i = 0; i <= n; i++)
{
for (w = 0; w <= W; w++)
{
if (i==0 || w==0)
K[i][w] = 0;
else if (wt[i-1] <= w)
K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
else
K[i][w] = K[i-1][w];
}
}
return K[n][W];
}
int main()
{
int i, n, val[20], wt[20], W;
printf("Enter number of items:");
scanf("%d", &n);
printf("Enter value and weight of items:\n");
for(i = 0;i < n; ++i){
scanf("%d%d", &val[i], &wt[i]);
}
printf("Enter size of knapsack:");
scanf("%d", &W);
printf("%d", knapSack(W, wt, val, n));
return 0;
}
```

22.SOLVE THE ALL PAIRS SHORTEST PATH PROBLEM.

```
#include<stdio.h>
int min(int,int);
void floyds(int p[10][10],int n)
{
int i,j,k;
for(k=1;k<=n;k++)
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(i==j)
p[i][j]=0;
```

```
else
p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
int min(int a,int b)
{
if(a<b)
return(a);
else
return(b);
}
void main()
{
int p[10][10],w,n,e,u,v,i,j;;
printf("\n Enter the number of vertices:");
scanf("%d",&n);
printf("\n Enter the number of edges:\n");
scanf("%d",&e);
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
p[i][j]=999;
}
for(i=1;i<=e;i++)
{
printf("\n Enter the end vertices of edge%d with its weight \n",i);
scanf("%d%d%d",&u,&v,&w);
p[u][v]=w;
}
printf("\n Matrix of input data:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
printf("%d \t",p[i][j]);
printf("\n");
}
floyds(p,n);
printf("\n Transitive closure:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
printf("%d \t",p[i][j]);
printf("\n");
}
printf("\n The shortest paths are:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
if(i!=j)
printf("\n <%d,%d>=%d",i,j,p[i][j]);
}
}
```

23.FIND THE OPTIMAL TOUR FOR TRAVELING SALES PERSON PROBLEM.

```
#include<stdio.h>
int a[10][10],visited[10],n,cost=0;
void get()
{
int i,j;
printf("Enter No. of Cities: ");
scanf("%d",&n);
printf("\nEnter Cost Matrix\n");
for(i=0;i < n;i++)
{
printf("\nEnter Elements of Row # : %d\n",i+1);
for( j=0;j < n;j++)
scanf("%d",&a[i][j]);
visited[i]=0;
}
printf("\n\nThe cost list is:\n\n");
for( i=0;i < n;i++)
{
printf("\n\n");
for(j=0;j < n;j++)
printf("\t%d",a[i][j]);
}
}
void mincost(int city)
{
int i,ncity;
visited[city]=1;
printf("%d -->",city+1);
ncity=least(city);
if(ncity==999)
{
ncity=0;
printf("%d",ncity+1);
cost+=a[city][ncity];
return;
}
mincost(ncity);
}
int least(int c)
{
int i,nc=999;
int min=999,kmin;
for(i=0;i < n;i++)
{
if((a[c][i]!=0)&&(visited[i]==0))
if(a[c][i] < min)
{
min=a[i][0]+a[c][i];
kmin=a[c][i];
nc=i;
}
}
if(min!=999)
cost+=kmin;
return nc;
}
void put()
{
printf("\n\nMinimum cost:");
printf("%d",cost);
}
void main()
{
clrscr();
get();
printf("\n\nThe Path is:\n\n");
mincost(0);
put();
}
```

24.N-QUEEN PEOBLEM:

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

int x[20];
int count;
void  print_board(int n)
{
int i,j;
printf("\n\nsolution %d:\n\n",++count);
for(i=1;i<=n;i++)
{
printf("\t%d",i);
}
for(i=1;i<=n;i++)
{
printf("\n\n%d",i);
for(j=1;j<=n;j++)
{
if(x[i]==j)
printf("\t Q");
else
printf("\t-");
}
}
printf("\npress any key to continue:");

}
int place(int row,int column)
```

```
{
int i;
for(i=1;i<=row-1;i++)
{
if(x[i]==column)
return 0;
else
if(abs(x[i]-column)==abs(i-row))
return 0;
}
return 1;
}
void queen(int row,int n)
{
int column;
for(column=1;column<=n;column++)
{
if(place(row,column))
{
x[row]=column;
if(row==n)
print_board(n);
else
queen(row+1,n);
}
}
}
void main()
```

```
{
int n,i,j;
printf("\n\t program for n-queens using backtracking:");
printf("\nenter no of queens");
scanf("%d",&n);
queen(1,n);
}
```

25.SOLVE SUM OF SUBSETS PROBLEM:

```
#include <stdio.h>
#include <stdlib.h>
#define ARRAYSIZE(a) (sizeof(a))/(sizeof(a[0]))
static int total_nodes;
void printSubset(int A[], int size)
{
for(int i = 0; i<size; i++)
{
printf("%*d", 5, A[i]);
}
printf("\n");
}
int comparator(const void *pLhs, const void *pRhs)
{
int *lhs = (int *)pLhs;
int *rhs = (int *)pRhs;
return *lhs > *rhs;
}
void subset_sum(int s[], int t[],ints_size, int t_size,int sum, int ite,int const target_sum)
{
```

```
total_nodes++;

if(target_sum == sum )

{

printSubset(t, t_size);

if(ite + 1<s_size && sum-s[ite]+s[ite+1] <= target_sum )

{

subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1,target_sum);

}

return;

}

else

{

if(ite<s_size && sum + s[ite] <= target_sum )

{

for( int i = ite; i<s_size; i++ )

{

t[t_size] = s[i];

if( sum + s[i] <= target_sum )

{

subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1,target_sum);

}

}

}

}

}


void generateSubsets(int s[], int size, int target_sum)

{
```

```
int *tuplet_vector = (int *)malloc(size * sizeof(int));
int total = 0;
qsort(s, size, sizeof(int), &comparator);
for( int i = 0; i<size; i++ )
{
total += s[i];
}
if( s[0] <= target_sum && total >= target_sum )
{ subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);
}
free(tuplet_vector);
}
int main()
{
int weights[] = {1,2,3,4,5,6,7,8,9};
int target = 7;
int size = ARRAYSIZE(weights);
generateSubsets(weights, size, target);
printf("Nodes generated %d\n", total_nodes);
return 0;
}
```

26. FIND THE CHROMATIC NUMBER OF THE GRAPH USING GRAPH COLORING.

```
#include<stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#define V 4
void printSolution(int color[]);
bool isSafe( bool graph[V][V], int color[])
```

```
{
for (int i = 0; i<V; i++)
for (int j = i + 1; j < V; j++)
if (graph[i][j] && color[j] == color[i])
return false;
return true;
}
bool graphColoring(bool graph[V][V], int m,int i, int color[V])
{
if (i == V)
{
if (isSafe(graph, color))
{
printSolution(color);
return true;
}
return false;
}
for (int j = 1; j<= m; j++)
{
color[i] = j;
if (graphColoring(graph, m, i + 1, color))
return true;
color[i] = 0;
}
return false;
}
 void printSolution(int color[])
```

```
{
printf("Solution Exists: \n Following are the assigned colors \n");
for (int i = 0; i<V; i++)
printf("%d ", color[i]);
printf("\n");
}
int main()
{
bool graph[V][V] = {

{ 0, 1, 1, 1 },
{ 1, 0, 1, 0 },
{ 1, 1, 0, 1 },
{ 1, 0, 1, 0 },

};
int m = 3;
int color[V];
for (int i = 0; i< V; i++)
color[i] = 0;
if (!graphColoring(graph, m, 0, color))
printf("Solution does not exist");
return 0;
}
```

27.FIND THE HAMILTONIAN CYCLE IN THE GRAPH.

```
#include<stdio.h>
#include <stdbool.h>
#include <stdlib.h>
```

```
#define V 5

void printSolution(int path[]);

bool isSafe(int v, bool graph[V][V], int path[], int pos)

{

if (graph [ path[pos-1] ][ v ] == 0)

return false;

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

if (path[i] == v)

return false;

return true;

}

bool hamCycleUtil(bool graph[V][V], int path[], int pos)

{

if (pos == V)

{

if ( graph[ path[pos-1] ][ path[0] ] == 1 )

return true;

else

return false;

}

for (int v = 1; v < V; v++)

{

if (isSafe(v, graph, path, pos))

{

path[pos] = v;

if (hamCycleUtil (graph, path, pos+1) == true)

return true;

path[pos] = -1;
}
}
return false;
}
bool hamCycle(bool graph[V][V])
{ int *path = malloc(V*sizeof(int));
for (int i = 0; i< V; i++)
path[i] = -1;
path[0] = 0;
if ( hamCycleUtil(graph, path, 1) == false )
{
printf("\nSolution does not exist");
return false;
}
printSolution(path);
return true;
}
void printSolution(int path[])
{
printf ( "Solution Exists:" " Following is one Hamiltonian Cycle \n");
for (int i = 0; i< V; i++)
printf(" %d ", path[i]);
printf(" %d " , path[0]);
printf("\n");
}
int main()
{
bool graph1[V][V] = {{0, 1, 0, 1, 0},
{1, 0, 1, 1, 1},

{0, 1, 0, 0, 1},

{1, 1, 0, 0, 1},

{0, 1, 1, 1, 0},

};

hamCycle(graph1);

bool graph2[V][V] = {

{0, 1, 0, 1, 0},

{1, 0, 1, 1, 1},

{0, 1, 0, 0, 1},

{1, 1, 0, 0, 0},

{0, 1, 1, 0, 0},

};

hamCycle(graph2);

return 0;

}
```

<u>28.0-1 KNAPSACK PROBLEM BY BRANCH BOUND METHOD:</u>

```
#include<stdio.h>

#define MAXSIZE 10

int Total_Items;

float Total_Weight,profit[MAXSIZE],weight[MAXSIZE];

struct tuple

{int flag , id;

float UB, LB;

}Left_Child,Right_Child,Current;


void Get_Input()

{
```

```
int i;
printf("Enter total number of items : ");
scanf("%d",&Total_Items);
printf("Enter the profit : \n");
for(i=1;i<=Total_Items;i++)
scanf("%f",&profit[i]);
printf("Enter the weight : \n");
for(i=1;i<=Total_Items;i++)
scanf("%f",&weight[i]);
printf("Enter weight constraint : ");
scanf("%f",&Total_Weight);
}

void Sort_Input()
{
float Ratio[Total_Items];
int i,j,temp;
for(i=1;i<=Total_Items;i++)
Ratio[i] = profit[i]/weight[i];
for(i=1;i<=Total_Items;i++)
{
for(j=i+1;j<=Total_Items;j++)
{
if(weight[j]<weight[j-1])
{
temp = profit[j];
profit[j] = profit[j-1];
profit[j-1] = temp;
temp = Ratio[j];

Ratio[j] = Ratio[j-1];

Ratio[j-1] = temp;

temp = weight[j];

weight[j] = weight[j-1];

weight[j-1] = temp;

}}}


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

{

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

{

if(Ratio[j]>Ratio[j-1])

{

temp = profit[j];

profit[j] = profit[j-1];

profit[j-1] = temp;

temp = Ratio[j];

Ratio[j] = Ratio[j-1];

Ratio[j-1] = temp;

temp = weight[j];

weight[j] = weight[j-1];

weight[j-1] = temp;

}}}

printf("Input is : \n");


rintf("Tuple\tProfit\tWeight\tRatio\n");

for(i=1;i<=Total_Items;i++)
```

```
printf("%d",i);
printf("\t");
printf("%f",profit[i]);
printf("\t");
printf("%f",weight[i]);
printf("\t");
printf("%f",Ratio[i]);
printf("\n");
}
float Calculate_UB(float Current_Wt,float Current_Pr,int Current_Item)
{
float cw = Current_Wt;
float cp = Current_Pr;
for(int i = Current_Item+1;i<=Total_Items;i++)
{
if(cw+weight[i]<=Total_Weight)
{
cw = cw + weight[i];
cp = cp - profit[i];
}}
return cp;
}
float Calculate_LB(float Current_Wt,float Current_Pr,int Current_Item)
{
float cw = Current_Wt;
float cp = Current_Pr;
for(int i = Current_Item+1;i<=Total_Items;i++)
{
```

```
cw = cw + weight[i];
if(cw<Total_Weight)
cp = cp - profit[i];
else
return (cp-(1-(cw-Total_Weight)/weight[i])*profit[i]);
}
return cp;
}
void Knapsack_BB()
{
int i,next,solution[MAXSIZE]={0};
float wt = 0,pr = 0;
Current.UB = Calculate_UB(0,0,0);
Current.LB = Calculate_LB(0,0,0);Current.flag = -1;
Current.id = 0;
i = 1;
do
{
next = Current.id+ 1;
Right_Child.UB = Calculate_UB(wt,pr,next);
Right_Child.LB = Calculate_LB(wt,pr,next);
Right_Child.flag = 0;
Right_Child.id = next;
Left_Child.flag = 1;
Left_Child.id = next;
if(wt + weight[next] <= Total_Weight)
{
Left_Child.UB =Calculate_UB(wt + weight[next],pr - profit[next],next);
Left_Child.LB =Calculate_LB(wt + weight[next],pr - profit[next],next);

}

else

{

Current.UB = pr;

Left_Child.LB = pr;

}

if(Left_Child.LB <= Right_Child.LB && Left_Child.UB <= Right_Child.UB)

Current = Left_Child;

else

Current = Right_Child;

solution[i] = Current.flag;

i++;

if(Current.flag == 1 )

{

pr = pr - profit[(Current.id)];

wt = wt + weight[(Current.id)];

}

}

while(Current.id != Total_Items);

printf("\nSolution = ");

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

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

printf("\t");

printf("\nMax Profit : ");

printf("%f",-(Current.LB));

printf("\n");

}
```

```
int main()
{
Get_Input();
Sort_Input();
Knapsack_BB();
return 0;
}
```

29.Branch and Bound Algorithm for Travelling Sales Person

```
 #include<stdio.h>
int a[10][10],visited[10],n,cost=0;
int least(int c);
void get1()
{
int i,j;
printf(" Enter No. of Cities: ");
scanf("%d",&n);
printf("\nEnter Cost Matrix: \n ");
for( i=0;i<n;i++)
{
printf("\n Enter Elements of Row # : %d\n",i+1);
for( j=0;j<n;j++)
scanf("%d",&a[i][j]);
visited[i]=0;
}
printf("\n\nThe cost list is:\n\n");
for( i=0;i<n;i++)
{
printf("\n\n");
```

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

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

}

}

void mincost(int city)

{

int i,ncity;

visited[city]=1;

printf("%d –>",city+1);

ncity=least(city);

if(ncity==999)

{

ncity=0;

printf("%d",ncity+1);

cost+=a[city][ncity];

return;

}

mincost(ncity);

}

int least(int c)

{

int i,nc=999;

int min=999,kmin;

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

{

if((a[c][i]!=0)&&(visited[i]==0))

if(a[c][i]<min)

{
```

```
min=a[i][0]+a[c][i];
kmin=a[c][i];
nc=i;
}
}
if(min!=999)
cost+=kmin;
return nc;
}
void put1()
{
printf("\n\nMinimum cost:");
printf("%d",cost);
}
void main()
{
get1();
printf("\n\nThe Path is:\n\n");
mincost(0);
put1();
}
```