**Program (Insertion sort)**

**#include <math.h>**

**#include <stdio.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 printArray(int arr[], int n){**

**int i;**

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

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

**printf("\n");**

**}**

**int main(){**

**int arr[] = {2,5,1,4,3};**

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

**insertionSort(arr, n);**

**printf("Sorted array: \n");**

**printArray(arr, n);**

**return 0;}**

**OUTPUT**



**Program (Bubble sort)**

**#include <stdio.h>**

**void swap(int\* xp, int\* yp){**

**int temp = \*xp;**

**\*xp = \*yp;**

**\*yp = temp;**

**}**

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

**int i, j;**

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

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

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

**swap(&arr[j], &arr[j + 1]);**

**}**

**void printArray(int arr[], int size){**

**int i;**

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

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

**printf("\n");**

**}**

**int main(){**

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

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

**bubbleSort(arr, n);**

**printf("Sorted array: \n");**

**printArray(arr, n);**

**return 0;**

**}**

**OUTPUT**



**Program (Recursive- Merge sort)**

**#include <stdio.h>**

**void mergeSort(int [], int, int, int);**

**void partition(int [],int, int);**

**int main(){**

**int list[50];**

**int i, size;**

**printf("Enter total number of elements:");**

**scanf("%d", &size);**

**printf("Enter the elements:\n");**

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

**scanf("%d", &list[i]);}**

**partition(list, 0, size - 1);**

**printf("After merge sort:\n");**

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

**printf("%d ",list[i]);}**

**return 0;}**

**void partition(int list[],int low,int high){**

**int mid;**

**if(low < high){**

**mid = (low + high) / 2;**

**partition(list, low, mid);**

**partition(list, mid + 1, high);**

**mergeSort(list, low, mid, high);**

**}}**

**void mergeSort(int list[],int low,int mid,int high){**

**int i, mi, k, lo, temp[50];**

**lo = low;**

**i = low;**

**mi = mid + 1;**

**while ((lo <= mid) && (mi <= high)){**

**if (list[lo] <= list[mi]){**

**temp[i] = list[lo];**

**lo++;}**

**else{**

**temp[i] = list[mi];**

**mi++;}**

**i++;}**

**if (lo > mid){**

**for (k = mi; k <= high; k++){**

**temp[i] = list[k];**

**i++;**

**}}**

**else{**

**for (k = lo; k <= mid; k++){**

**temp[i] = list[k];**

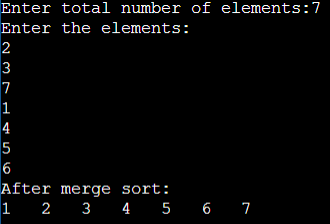
**i++;**

**}}**

**for (k = low; k <= high; k++){**

**list[k] = temp[k];**

**}}**

**OUTPUT**

**Program (Recursive- Linear Search)**

**#include <stdio.h>**

**int RecursiveLS(int arr[], int value, int index, int n){**

**int pos = 0;**

**if(index >= n){**

**return 0;}**

**else if (arr[index] == value){**

**pos = index + 1;**

**return pos;}**

**else{**

**return RecursiveLS(arr, value, index+1, n);}**

**return pos;}**

**int main(){**

**int n, value, pos, m = 0, arr[100];**

**printf("Enter the total elements in the array ");**

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

**printf("Enter the array elements\n");**

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

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

**printf("Enter the element to search ");**

**scanf("%d", &value);**

**pos = RecursiveLS(arr, value, 0, n);**

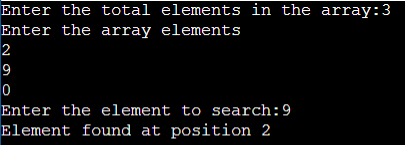
**if (pos != 0){**

**printf("Element found at pos %d ", pos);}**

**else{**

**printf("Element not found");}**

**return 0;}**

**OUTPUT**

**Program (Quick sort)**

**#include <stdio.h>**

**void quicksort (int [], int, int);**

**int main(){**

**int list[50];**

**int size, i;**

**printf("Enter the number of elements: ");**

**scanf("%d", &size);**

**printf("Enter the elements to be sorted:\n");**

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

**scanf("%d", &list[i]);}**

**quicksort(list, 0, size - 1);**

**printf("After applying quick sort\n");**

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

**printf("%d ", list[i]);}**

**printf("\n");**

**return 0;}**

**void quicksort(int list[], int low, int high){**

**int pivot, i, j, temp;**

**if (low < high){**

**pivot = low;**

**i = low;**

**j = high;**

**while (i < j) {**

**while (list[i] <= list[pivot] && i <= high){**

**i++;}**

**while (list[j] > list[pivot] && j >= low){**

**j--;}**

**if (i < j){**

**temp = list[i];**

**list[i] = list[j];**

**list[j] = temp;}}**

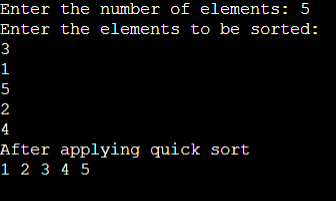
**temp = list[j];**

**list[j] = list[pivot];**

**list[pivot] = temp;**

**quicksort(list, low, j - 1);**

**quicksort(list, j + 1, high);}}**

**OUTPUT**

**Program (Binary search)**

**#include <stdio.h>**

**int iterativeBinarySearch(int array[], int start\_index, int end\_index, int element){**

**while (start\_index <= end\_index){**

**int middle = start\_index + (end\_index- start\_index )/2;**

**if (array[middle] == element)**

**return middle;**

**if (array[middle] < element)**

**start\_index = middle + 1;**

**else**

**end\_index = middle - 1;}**

**return -1;}**

**int main(void){**

**int array[] = {1, 4, 7, 9, 16, 56, 70};**

**int n = 7;**

**int element = 16;**

**int found\_index = iterativeBinarySearch(array, 0, n-1, element);**

**if(found\_index == -1 ) {**

**printf("Element not found in the array ");}**

**else {**

**printf("Element found at index : %d",found\_index);}**

**return 0;}**

**OUTPUT**

**Program (Strassen Matrix Multiplication)**

**#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 \n");**

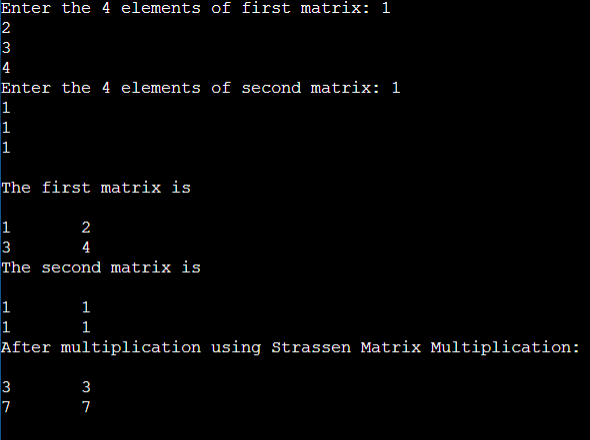
**for(i=0;i<2;i++){**

**printf("\n");**

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

**printf("%d\t",c[i][j]);}**

**return 0;}**

**OUTPUT**

**Program (Max & Min in an array)**

**#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("Enter elements in the array: ");**

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

**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];**

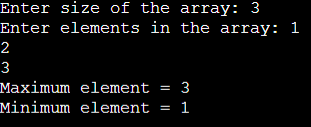
**}}**

**printf("Maximum element = %d\n", max);**

**printf("Minimum element = %d", min);**

**return 0;**

**}**

**OUTPUT**

**Program (Convex Hull Problem)**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <math.h>**

**typedef struct point {**

**double x;**

**double y;**

**}POINT,VECTOR;**

**POINT b[1000];**

**VECTOR normal;**

**int n;**

**int upper\_lower(int i, VECTOR ab, double c) {**

**double x, y,result;**

**y = b[i].y;**

**x = normal.x\*b[i].x;**

**result = -(x + c) / normal.y;**

**if (y>result) return 1;**

**if (y == result) return 0;**

**else**

**return -1; }**

**int ccw(VECTOR v,VECTOR v2) {**

**double cp;**

**cp = v2.x\*v.y - v2.y\*v.x;**

**if (cp == abs(cp)) return 1;**

**else**

**return -1; }**

**double vector\_length(VECTOR v) {**

**return sqrt(pow(v.x, 2) + pow(v.y, 2)); }**

**int cmp\_points(const void \*p1, const void \*p2) {**

**const POINT \*pt1 = p1;**

**const POINT \*pt2 = p2;**

**if (pt1->x > pt2->x)**

**return 1;**

**if (pt1->x < pt2->x)**

**return -1;**

**if (pt1->y > pt2->y)**

**return 1;**

**if (pt1->y < pt2->y)**

**return -1;**

**return 0; }**

**int main() {**

**int i,poloha,upper[1000],lower[1000],h=0,d=0;**

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

**if (n <= 0 && n > 1000) return 0;**

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

**scanf("%lf %lf", &b[i].x, &b[i].y); }**

**qsort(b, n, sizeof(POINT), cmp\_points);**

**VECTOR ab;**

**double c;**

**ab.x = b[n - 1].x - b[0].x;**

**ab.y = b[n - 1].y - b[0].y;**

**normal.x = -ab.y;**

**normal.y = ab.x;**

**c = -normal.x\*b[0].x - (normal.y\*b[0].y);**

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

**poloha = upper\_lower(i,ab,c);**

**if (poloha == 1) upper[h++] = i;**

**if (poloha == -1) lower[d++]=i;**

**if (poloha == 0) {**

**upper[h++] = i;**

**lower[d++] = i; } }**

**int j = 0;**

**double v, length = 0;**

**VECTOR v1, v2, v3,v4;**

**v3.x = 0; v3.y = 0;**

**for (i = 0; ; i++) {**

**int in = 0;**

**if (lower[i + 2] < 0) {**

**v1.x = b[lower[i + 1]].x - b[lower[0]].x;**

**v1.y = b[lower[i + 1]].y - b[lower[0]].y;**

**v2.x = b[lower[i]].x - b[lower[i + 1]].x;**

**v2.y = b[lower[i]].y - b[lower[i + 1]].y;**

**length += vector\_length(v1);**

**length += vector\_length(v2);**

**break; }**

**v1.x = b[lower[i + 1]].x - b[lower[i]].x;**

**v1.y = b[lower[i + 1]].y - b[lower[i]].y;**

**v2.x = b[lower[i + 2]].x - b[lower[i]].x;**

**v2.y = b[lower[i + 2]].y - b[lower[i]].y;**

**in = ccw(v1, v2);**

**if (in == 1) {**

**length += vector\_length(v1);**

**v3 = v2;**

**v4 = v1; }**

**if (in == -1) {**

**length -= vector\_length(v4);**

**if (v3.x != 0 && v3.y != 0) {**

**length += vector\_length(v3);**

**v3.x = 0; v3.y = 0; }**

**else {**

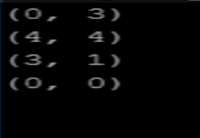
**length += vector\_length(v2);**

**} } }**

**printf("%.3lf", length);**

**return 0; }**

**OUTPUT**



**Program (Huffman coding)**

**#include<string.h>**

**#include<stdio.h>**

**#include<stdlib.h>**

**typedef struct node**

**{**

**char ch;**

**int freq;**

**struct node \*left;**

**struct node \*right;**

**}node;**

**node \* heap[100];**

**int heapSize=0;**

**void Insert(node \* element)**

**{**

**heapSize++;**

**heap[heapSize] = element;**

**int now = heapSize;**

**while(heap[now/2] -> freq > element -> freq)**

**{**

**heap[now] = heap[now/2];**

**now /= 2;**

**}**

**heap[now] = element;**

**}**

**node \* DeleteMin()**

**{**

**node \* minElement,\*lastElement;**

**int child,now;**

**minElement = heap[1];**

**lastElement = heap[heapSize--];**

**for(now = 1; now\*2 <= heapSize ;now = child)**

**{**

**child = now\*2;**

**if(child != heapSize && heap[child+1]->freq < heap[child] -> freq )**

**{**

**child++;**

**}**

**if(lastElement -> freq > heap[child] -> freq)**

**{**

**heap[now] = heap[child];**

**}**

**else**

**{**

**break;**

**}**

**}**

**heap[now] = lastElement;**

**return minElement;**

**}**

**void print(node \*temp,char \*code)**

**{**

**if(temp->left==NULL && temp->right==NULL)**

**{**

**printf("char %c code %s\n",temp->ch,code);**

**return;**

**}**

**int length = strlen(code);**

**char leftcode[10],rightcode[10];**

**strcpy(leftcode,code);**

**strcpy(rightcode,code);**

**leftcode[length] = '0';**

**leftcode[length+1] = '\0';**

**rightcode[length] = '1';**

**rightcode[length+1] = '\0';**

**print(temp->left,leftcode);**

**print(temp->right,rightcode);**

**}**

**int main()**

**{**

**heap[0] = (node \*)malloc(sizeof(node));**

**heap[0]->freq = 0;**

**int n ;**

**printf("Enter the no of characters: ");**

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

**printf("Enter the characters and their frequencies: ");**

**char ch;**

**int freq,i;**

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

**{**

**scanf(" %c",&ch);**

**scanf("%d",&freq);**

**node \* temp = (node \*) malloc(sizeof(node));**

**temp -> ch = ch;**

**temp -> freq = freq;**

**temp -> left = temp -> right = NULL;**

**Insert(temp);**

**}**

**if(n==1)**

**{**

**printf("char %c code 0\n",ch);**

**return 0;**

**}**

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

**{**

**node \* left = DeleteMin();**

**node \* right = DeleteMin();**

**node \* temp = (node \*) malloc(sizeof(node));**

**temp -> ch = 0;**

**temp -> left = left;**

**temp -> right = right;**

**temp -> freq = left->freq + right -> freq;**

**Insert(temp);**

**}**

**node \*tree = DeleteMin();**

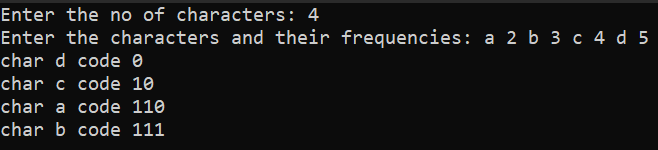
**char code[10];**

**code[0] = '\0';**

**print(tree,code);**

**}**

**OUTPUT**

****

**Program (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;**

**}**

**}**

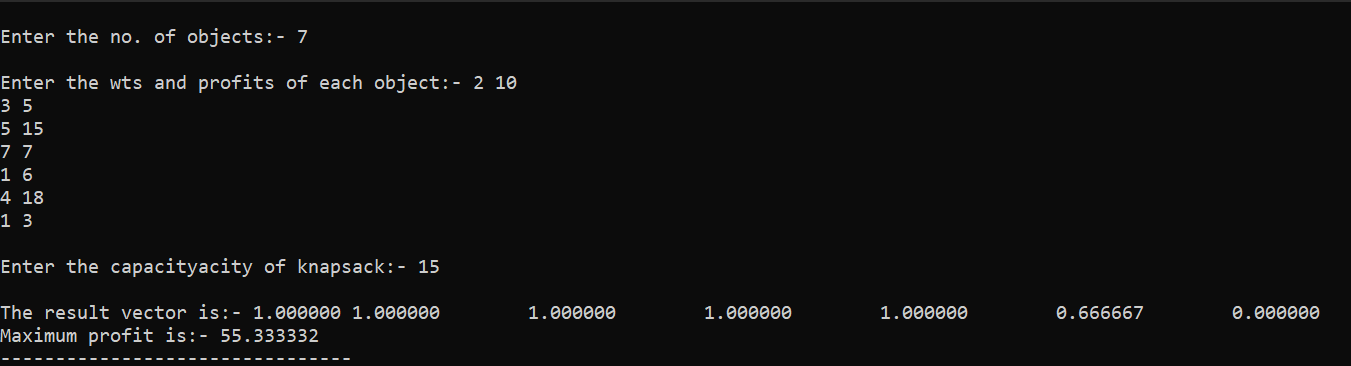
**}**

**knapsack(num, weight, profit, capacity);**

**return(0);**

**}**

**OUTPUT**

****

**Program (Various Tree Traversals)**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct node {**

**int data;**

**struct node \*leftChild;**

**struct node \*rightChild;**

**};**

**struct node \*root = NULL;**

**void insert(int data) {**

**struct node \*tempNode = (struct node\*) malloc(sizeof(struct node));**

**struct node \*current;**

**struct node \*parent;**

**tempNode->data = data;**

**tempNode->leftChild = NULL;**

**tempNode->rightChild = NULL;**

**//if tree is empty**

**if(root == NULL) {**

**root = tempNode;**

**} else {**

**current = root;**

**parent = NULL;**

**while(1) {**

**parent = current;**

**//go to left of the tree**

**if(data < parent->data) {**

**current = current->leftChild;**

**//insert to the left**

**if(current == NULL) {**

**parent->leftChild = tempNode;**

**return;**

**}**

**} //go to right of the tree**

**else {**

**current = current->rightChild;**

**//insert to the right**

**if(current == NULL) {**

**parent->rightChild = tempNode;**

**return;**

**}**

**}**

**}**

**}**

**}**

**struct node\* search(int data) {**

**struct node \*current = root;**

**printf("Visiting elements: ");**

**while(current->data != data) {**

**if(current != NULL)**

**printf("%d ",current->data);**

**//go to left tree**

**if(current->data > data) {**

**current = current->leftChild;**

**}**

**//else go to right tree**

**else {**

**current = current->rightChild;**

**}**

**//not found**

**if(current == NULL) {**

**return NULL;**

**}**

**}**

**return current;**

**}**

**void pre\_order\_traversal(struct node\* root) {**

**if(root != NULL) {**

**printf("%d ",root->data);**

**pre\_order\_traversal(root->leftChild);**

**pre\_order\_traversal(root->rightChild);**

**}**

**}**

**void inorder\_traversal(struct node\* root) {**

**if(root != NULL) {**

**inorder\_traversal(root->leftChild);**

**printf("%d ",root->data);**

**inorder\_traversal(root->rightChild);**

**}**

**}**

**void post\_order\_traversal(struct node\* root) {**

**if(root != NULL) {**

**post\_order\_traversal(root->leftChild);**

**post\_order\_traversal(root->rightChild);**

**printf("%d ", root->data);**

**}**

**}**

**int main() {**

**int i;**

**int array[7] = { 27, 14, 35, 10, 19, 31, 42 };**

**for(i = 0; i < 7; i++)**

**insert(array[i]);**

**i = 31;**

**struct node \* temp = search(i);**

**if(temp != NULL) {**

**printf("[%d] Element found.", temp->data);**

**printf("\n");**

**}else {**

**printf("[ x ] Element not found (%d).\n", i);**

**}**

**i = 15;**

**temp = search(i);**

**if(temp != NULL) {**

**printf("[%d] Element found.", temp->data);**

**printf("\n");**

**}else {**

**printf("[ x ] Element not found (%d).\n", i);**

**}**

**printf("\nPreorder traversal: ");**

**pre\_order\_traversal(root);**

**printf("\nInorder traversal: ");**

**inorder\_traversal(root);**

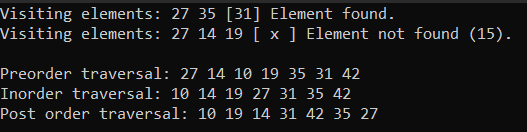
**printf("\nPost order traversal: ");**

**post\_order\_traversal(root);**

**return 0;**

**}**

**OUTPUT**

****

**Program (Kruskal’s Algorithm)**

**#include <stdio.h>**

**#define MAX 30**

**typedef struct edge {**

**int u, v, w;**

**} edge;**

**typedef struct edge\_list {**

**edge data[MAX];**

**int n;**

**} edge\_list;**

**edge\_list elist;**

**int Graph[MAX][MAX], n;**

**edge\_list spanlist;**

**void kruskalAlgo();**

**int find(int belongs[], int vertexno);**

**void applyUnion(int belongs[], int c1, int c2);**

**void sort();**

**void print();**

**// Applying Krushkal Algo**

**void kruskalAlgo() {**

**int belongs[MAX], i, j, cno1, cno2;**

**elist.n = 0;**

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

**for (j = 0; j < i; j++) {**

**if (Graph[i][j] != 0) {**

**elist.data[elist.n].u = i;**

**elist.data[elist.n].v = j;**

**elist.data[elist.n].w = Graph[i][j];**

**elist.n++;**

**}**

**}**

**sort();**

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

**belongs[i] = i;**

**spanlist.n = 0;**

**for (i = 0; i < elist.n; i++) {**

**cno1 = find(belongs, elist.data[i].u);**

**cno2 = find(belongs, elist.data[i].v);**

**if (cno1 != cno2) {**

**spanlist.data[spanlist.n] = elist.data[i];**

**spanlist.n = spanlist.n + 1;**

**applyUnion(belongs, cno1, cno2);**

**}**

**}**

**}**

**int find(int belongs[], int vertexno) {**

**return (belongs[vertexno]);**

**}**

**void applyUnion(int belongs[], int c1, int c2) {**

**int i;**

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

**if (belongs[i] == c2)**

**belongs[i] = c1;**

**}**

**// Sorting algo**

**void sort() {**

**int i, j;**

**edge temp;**

**for (i = 1; i < elist.n; i++)**

**for (j = 0; j < elist.n - 1; j++)**

**if (elist.data[j].w > elist.data[j + 1].w) {**

**temp = elist.data[j];**

**elist.data[j] = elist.data[j + 1];**

**elist.data[j + 1] = temp;**

**}**

**}**

**// Printing the result**

**void print() {**

**int i, cost = 0;**

**for (i = 0; i < spanlist.n; i++) {**

**printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);**

**cost = cost + spanlist.data[i].w;**

**}**

**printf("\nSpanning tree cost: %d", cost);**

**}**

**int main() {**

**int i, j, total\_cost;**

**n = 6;**

**Graph[0][0] = 0;**

**Graph[0][1] = 4;**

**Graph[0][2] = 4;**

**Graph[0][3] = 0;**

**Graph[0][4] = 0;**

**Graph[0][5] = 0;**

**Graph[0][6] = 0;**

**Graph[1][0] = 4;**

**Graph[1][1] = 0;**

**Graph[1][2] = 2;**

**Graph[1][3] = 0;**

**Graph[1][4] = 0;**

**Graph[1][5] = 0;**

**Graph[1][6] = 0;**

**Graph[2][0] = 4;**

**Graph[2][1] = 2;**

**Graph[2][2] = 0;**

**Graph[2][3] = 3;**

**Graph[2][4] = 4;**

**Graph[2][5] = 0;**

**Graph[2][6] = 0;**

**Graph[3][0] = 0;**

**Graph[3][1] = 0;**

**Graph[3][2] = 3;**

**Graph[3][3] = 0;**

**Graph[3][4] = 3;**

**Graph[3][5] = 0;**

**Graph[3][6] = 0;**

**Graph[4][0] = 0;**

**Graph[4][1] = 0;**

**Graph[4][2] = 4;**

**Graph[4][3] = 3;**

**Graph[4][4] = 0;**

**Graph[4][5] = 0;**

**Graph[4][6] = 0;**

**Graph[5][0] = 0;**

**Graph[5][1] = 0;**

**Graph[5][2] = 2;**

**Graph[5][3] = 0;**

**Graph[5][4] = 3;**

**Graph[5][5] = 0;**

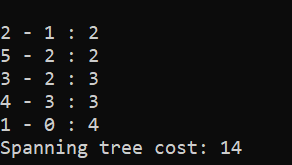
**Graph[5][6] = 0;**

**kruskalAlgo();**

**print();**

**}**

**OUTPUT**

****

**Program (Longest common sequence)**

**#include <stdio.h>**

**#include <string.h>**

**int i, j, m, n, LCS\_table[20][20];**

**char S1[20] = "abaaba", S2[20] = "babbab", b[20][20];**

**void lcsAlgo() {**

**m = strlen(S1);**

**n = strlen(S2);**

**// Filling 0's in the matrix**

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

**LCS\_table[i][0] = 0;**

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

**LCS\_table[0][i] = 0;**

**// Creating the mtrix in bottom-up way**

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

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

**if (S1[i - 1] == S2[j - 1]) {**

**LCS\_table[i][j] = LCS\_table[i - 1][j - 1] + 1;**

**} else if (LCS\_table[i - 1][j] >= LCS\_table[i][j - 1]) {**

**LCS\_table[i][j] = LCS\_table[i - 1][j];**

**} else {**

**LCS\_table[i][j] = LCS\_table[i][j - 1];**

**}**

**}**

**int index = LCS\_table[m][n];**

**char lcsAlgo[index + 1];**

**lcsAlgo[index] = '\0';**

**int i = m, j = n;**

**while (i > 0 && j > 0) {**

**if (S1[i - 1] == S2[j - 1]) {**

**lcsAlgo[index - 1] = S1[i - 1];**

**i--;**

**j--;**

**index--;**

**}**

**else if (LCS\_table[i - 1][j] > LCS\_table[i][j - 1])**

**i--;**

**else**

**j--;**

**}**

**// Printing the sub sequences**

**printf("S1 : %s \nS2 : %s \n", S1, S2);**

**printf("LCS: %s", lcsAlgo);**

**}**

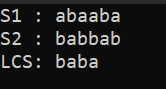
**int main() {**

**lcsAlgo();**

**printf("\n");**

**}**

**OUTPUT**



**Program (N queen’s problem)**

**#include<stdio.h>**

**#include<conio.h>**

**#include<math.h>**

**int a[30],count=0;**

**int place(int pos) {**

**int i;**

**for (i=1;i<pos;i++) {**

**if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))**

**return 0;**

**}**

**return 1;**

**}**

**void print\_sol(int n) {**

**int i,j;**

**count++;**

**printf("\n\nSolution #%d:\n",count);**

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

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

**if(a[i]==j)**

**printf("Q\t"); else**

**printf("\*\t");**

**}**

**printf("\n");**

**}**

**}**

**void queen(int n) {**

**int k=1;**

**a[k]=0;**

**while(k!=0) {**

**a[k]=a[k]+1;**

**while((a[k]<=n)&&!place(k))**

**a[k]++;**

**if(a[k]<=n) {**

**if(k==n)**

**print\_sol(n); else {**

**k++;**

**a[k]=0;**

**}**

**} else**

**k--;**

**}**

**}**

**void main() {**

**int i,n;**

**clrscr();**

**printf("Enter the number of Queens\n");**

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

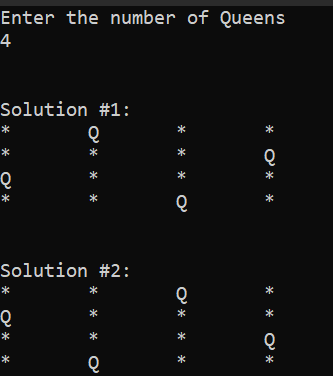
**queen(n);**

**printf("\nTotal solutions=%d",count);**

**getch();**

**}**

**OUTPUT**

****

**Program (Travelling salesman problem)**

**#include <stdio.h>**

**int matrix[25][25], visited\_cities[10], limit, cost = 0;**

**int tsp(int c)**

**{**

**int count, nearest\_city = 999;**

**int minimum = 999, temp;**

**for(count = 0; count < limit; count++)**

**{**

**if((matrix[c][count] != 0) && (visited\_cities[count] == 0))**

**{**

**if(matrix[c][count] < minimum)**

**{**

**minimum = matrix[count][0] + matrix[c][count];**

**}**

**temp = matrix[c][count];**

**nearest\_city = count;**

**}**

**}**

**if(minimum != 999)**

**{**

**cost = cost + temp;**

**}**

**return nearest\_city;**

**}**

**void minimum\_cost(int city)**

**{**

**int nearest\_city;**

**visited\_cities[city] = 1;**

**printf("%d ", city + 1);**

**nearest\_city = tsp(city);**

**if(nearest\_city == 999)**

**{**

**nearest\_city = 0;**

**printf("%d", nearest\_city + 1);**

**cost = cost + matrix[city][nearest\_city];**

**return;**

**}**

**minimum\_cost(nearest\_city);**

**}**

**int main()**

**{**

**int i, j;**

**printf("Enter Total Number of Cities:\t");**

**scanf("%d", &limit);**

**printf("\nEnter Cost Matrix\n");**

**for(i = 0; i < limit; i++)**

**{**

**printf("\nEnter %d Elements in Row[%d]\n", limit, i + 1);**

**for(j = 0; j < limit; j++)**

**{**

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

**}**

**visited\_cities[i] = 0;**

**}**

**printf("\nEntered Cost Matrix\n");**

**for(i = 0; i < limit; i++)**

**{**

**printf("\n");**

**for(j = 0; j < limit; j++)**

**{**

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

**}**

**}**

**printf("\n\nPath:\t");**

**minimum\_cost(0);**

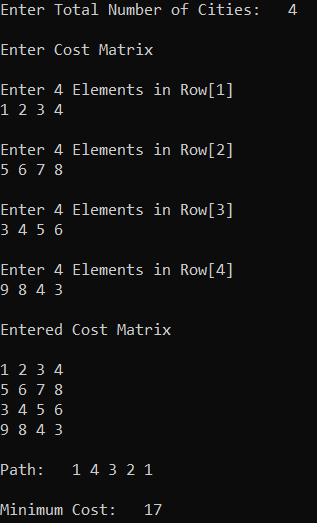
**printf("\n\nMinimum Cost: \t");**

**printf("%d\n", cost);**

**return 0;**

**}**

**OUTPUT**

****

**Program (BFS)**

**#include<stdio.h>**

**#include<stdlib.h>**

**#define MAX 100**

**#define initial 1**

**#define waiting 2**

**#define visited 3**

**int n;**

**int adj[MAX][MAX];**

**int state[MAX];**

**void create\_graph();**

**void BF\_Traversal();**

**void BFS(int v);**

**int queue[MAX], front = -1,rear = -1;**

**void insert\_queue(int vertex);**

**int delete\_queue();**

**int isEmpty\_queue();**

**int main()**

**{**

**create\_graph();**

**BF\_Traversal();**

**return 0;**

**}**

**void BF\_Traversal()**

**{**

**int v;**

**for(v=0; v<n; v++)**

**state[v] = initial;**

**printf("Enter Start Vertex for BFS: \n");**

**scanf("%d", &v);**

**BFS(v);**

**}**

**void BFS(int v)**

**{**

**int i;**

**insert\_queue(v);**

**state[v] = waiting;**

**while(!isEmpty\_queue())**

**{**

**v = delete\_queue( );**

**printf("%d ",v);**

**state[v] = visited;**

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

**{**

**if(adj[v][i] == 1 && state[i] == initial)**

**{**

**insert\_queue(i);**

**state[i] = waiting;**

**}**

**}**

**}**

**printf("\n");**

**}**

**void insert\_queue(int vertex)**

**{**

**if(rear == MAX-1)**

**printf("Queue Overflow\n");**

**else**

**{**

**if(front == -1)**

**front = 0;**

**rear = rear+1;**

**queue[rear] = vertex ;**

**}**

**}**

**int isEmpty\_queue()**

**{**

**if(front == -1 || front > rear)**

**return 1;**

**else**

**return 0;**

**}**

**int delete\_queue()**

**{**

**int delete\_item;**

**if(front == -1 || front > rear)**

**{**

**printf("Queue Underflow\n");**

**exit(1);**

**}**

**delete\_item = queue[front];**

**front = front+1;**

**return delete\_item;**

**}**

**void create\_graph()**

**{**

**int count,max\_edge,origin,destin;**

**printf("Enter number of vertices : ");**

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

**max\_edge = n\*(n-1);**

**for(count=1; count<=max\_edge; count++)**

**{**

**printf("Enter edge %d( -1 -1 to quit ) : ",count);**

**scanf("%d %d",&origin,&destin);**

**if((origin == -1) && (destin == -1))**

**break;**

**if(origin>=n || destin>=n || origin<0 || destin<0)**

**{**

**printf("Invalid edge!\n");**

**count--;**

**}**

**else**

**{**

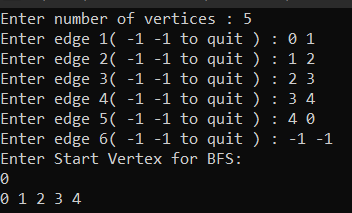
**adj[origin][destin] = 1;**

**}**

**}**

**}**

**OUTPUT**

****

**Program (DFS)**

**#include<stdio.h>**

**void DFS(int);**

**int G[10][10],visited[10],n; //n is no of vertices and graph is sorted in array G[10][10]**

**int main()**

**{**

**int i,j;**

**printf("Enter number of vertices:");**

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

**//read the adjecency matrix**

**printf("\nEnter adjecency matrix of the graph:");**

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

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

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

**//visited is initialized to zero**

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

**visited[i]=0;**

**DFS(0);**

**}**

**void DFS(int i)**

**{**

**int j;**

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

**visited[i]=1;**

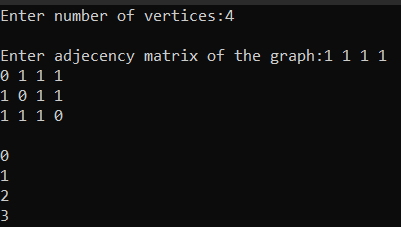
**for(j=0;j<n;j++)**

**if(!visited[j]&&G[i][j]==1)**

**DFS(j);**

**}**

**OUTPUT**

****

**Program (Randomized QuikSort)**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <time.h>**

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

**{**

**int pivot = arr[low];**

**int i = low - 1, j = high + 1;**

**while (1) {**

**do {**

**i++;**

**} while (arr[i] < pivot);**

**do {**

**j--;**

**} while (arr[j] > pivot);**

**if (i >= j)**

**return j;**

**int temp = arr[i];**

**arr[i] = arr[j];**

**arr[j] = temp;**

**}**

**}**

**int partition\_r(int arr[], int low, int high)**

**{**

**srand(time(0));**

**int random = low + rand() % (high - low);**

**int temp = arr[random];**

**arr[random] = arr[low];**

**arr[low] = temp;**

**return partition(arr, low, high);**

**}**

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

**{**

**if (low < high) {**

**int pi = partition\_r(arr, low, high);**

**quickSort(arr, low, pi);**

**quickSort(arr, pi + 1, high);**

**}**

**}**

**void printArray(int arr[], int n)**

**{**

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

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

**printf("\n");**

**}**

**int main()**

**{**

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

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

**quickSort(arr, 0, n - 1);**

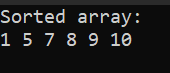
**printf("Sorted array: \n");**

**printArray(arr, n);**

**return 0;**

**}**

**OUTPUT**



**Program (String Matching Algorithm)**

**#include <stdio.h>**

**#include <string.h>**

**int match(char [], char []);**

**int main() {**

**char a[100], b[100];**

**int position;**

**printf("Enter some text\n");**

**gets(a);**

**printf("Enter a string to find\n");**

**gets(b);**

**position = match(a, b);**

**if (position != -1) {**

**printf("Found at location: %d\n", position + 1);**

**}**

**else {**

**printf("Not found.\n");**

**}**

**return 0;**

**}**

**int match(char text[], char pattern[]) {**

**int c, d, e, text\_length, pattern\_length, position = -1;**

**text\_length = strlen(text);**

**pattern\_length = strlen(pattern);**

**if (pattern\_length > text\_length) {**

**return -1;**

**}**

**for (c = 0; c <= text\_length - pattern\_length; c++) {**

**position = e = c;**

**for (d = 0; d < pattern\_length; d++) {**

**if (pattern[d] == text[e]) {**

**e++;**

**}**

**else {**

**break;**

**}**

**}**

**if (d == pattern\_length) {**

**return position;**

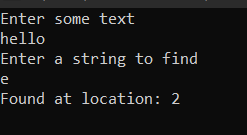
**}**

**}**

**return -1;**

**}**

**OUTPUT**

****

**Program (Analyzing a real time problem)**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <sys/time.h>**

**#include <omp.h>**

**void simplemerge(int a[], int low, int mid, int high)**

**{**

**int i,j,k,c[20000];**

**i=low;**

**j=mid+1;**

**k=low;**

**int tid;**

**omp\_set\_num\_threads(10);**

**{**

**tid=omp\_get\_thread\_num();**

**while(i<=mid&&j<=high)**

**{**

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

**{**

**c[k]=a[i];**

**//printf("%d%d",tid,c[k]);**

**i++;**

**k++;**

**}**

**else**

**{**

**c[k]=a[j];**

**//printf("%d%d", tid, c[k]);**

**j++;**

**k++;**

**}**

**}**

**}**

**while(i<=mid)**

**{**

**c[k]=a[i];**

**i++;**

**k++;**

**}**

**while(j<=high)**

**{**

**c[k]=a[j];**

**j++;**

**k++;**

**}**

**for(k=low;k<=high;k++)**

**a[k]=c[k];**

**}**

**void merge(int a[],int low,int high)**

**{**

**int mid;**

**if(low < high)**

**{**

**mid=(low+high)/2;**

**merge(a,low,mid);**

**merge(a,mid+1,high);**

**simplemerge(a,low,mid,high);**

**}**

**}**

**void getnumber(int a[], int n)**

**{**

**int i;**

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

**a[i]=rand()%100;**

**}**

**int main()**

**{**

**FILE \*fp;**

**int a[2000],i;**

**struct timeval tv;**

**double start, end, elapse;**

**fp=fopen("mergesort.txt","w");**

**for(i=10;i<=1000;i+=10)**

**{**

**getnumber(a,i);**

**gettimeofday(&tv,NULL);**

**start=tv.tv\_sec+(tv.tv\_usec/1000000.0);**

**merge(a,0,i-1);**

**gettimeofday(&tv,NULL);**

**end=tv.tv\_sec+(tv.tv\_usec/1000000.0);**

**elapse=end-start;**

**fprintf(fp,"%d\t%lf\n",i,elapse);**

**}**

**fclose(fp);**

**system("gnuplot");**

**return 0;**

**}**

**mergesort.gpl**

**Gnuplot script file for plotting data in file "mergesort.txt" This file is called mergesort.gpl**

**set terminal png font arial**

**set title "Time Complexity for Merge Sort"**

**set autoscale**

**set xlabel "Size of Input"**

**set ylabel "Sorting Time (microseconds)"**

**set grid**

**set output "mergesort.png"**

**plot "mergesort.txt" t "Merge Sort" with lines**

**OUTPUT**

