**Program 1**

**Program for recursive linear search**

#include<stdio.h>

int linearSearch(int arr[], int l, int r, int x)

{

if (r < l)

return -1;

if (arr[l] == x)

return l;

if (arr[r] == x)

return r;

return linearSearch(arr, l+1, r-1, x);

}

int main()

{

int arr[10], i;

int n;

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

scanf("%d",&n);

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

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

{

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

}

int key;

printf("Enter the key you want to find\n");

scanf("%d",&key);

int index = linearSearch(arr, 0, n, key);

if (index != -1)

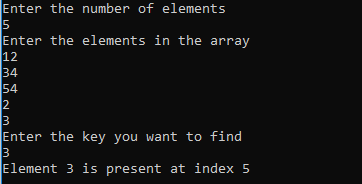
printf("Element %d is present at index %d", key, index);

else

printf("Element %d is not present", key);

return 0;

}



**Program 2**

**Program for binary search**

#include<stdio.h>

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

{

if (r >= l)

{

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

if (arr[mid] == x) return mid;

if (arr[mid] > x) return binarySearch(arr, l, mid-1, x);

return binarySearch(arr, mid+1, r, x);

}

return -1;

}

int main(void)

{

int arr[10],i,n,key;

printf("Enter the number of elements in an array\n");

scanf("%d",&n);

printf("Enter the elements in an array\n");

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

{

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

}

printf("Enter the key you want to find\n");

scanf("%d",&key);

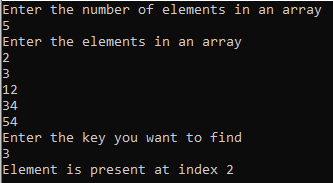
int result = binarySearch(arr, 0, n-1, key);

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

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

return 0;

}



**Program 3**

**Program for Heap sort**

#include <stdio.h>

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

{

int largest = i;

int l = 2\*i + 1;

int r = 2\*i + 2;

int temp;

if (l < n && arr[l] > arr[largest])

largest = l;

if (r < n && arr[r] > arr[largest])

largest = r;

if (largest != i)

{

temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

heapify(arr, n, largest);

} }

void heapSort(int arr[], int n)

{

int temp;

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

heapify(arr, n, i);

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

{

temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

}

void printArray(int arr[], int n)

{

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

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

printf("\n");

}

int main()

{

int arr[10], i, n;

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

scanf("%d",&n);

printf("Enter the elements\n");

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

{

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

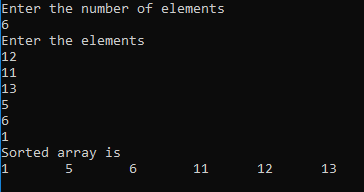
}

heapSort(arr, n);

printf("Sorted array is\n");

printArray(arr, n);

}



**Program 4**

**Program for Merge Sort**

#include<stdlib.h>

#include<stdio.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

{

arr[k] = R[j];

j++;

}

k++;

}

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

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

printf("Given array is \n");

printArray(arr, arr\_size);

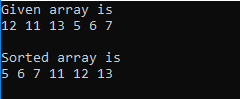
mergeSort(arr, 0, arr\_size - 1);

printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0;

}



**Program 5**

**Program for Selection Sort**

#include <stdio.h>

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

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

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

{

min\_idx = i;

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

if (arr[j] < arr[min\_idx])

min\_idx = j;

swap(&arr[min\_idx], &arr[i]);

} }

void printArray(int arr[], int size)

{

int i;

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

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

printf("\n");

}

int main()

{

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

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

selectionSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}



**Program 6**

**Program for 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[] = { 12, 11, 13, 5, 6 };

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

insertionSort(arr, n);

printArray(arr, n);

return 0;

}



**Program 7**

**Program for Quick sort**

#include<stdio.h>

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

{

int t = \*a;

\*a = \*b;

\*b = t;

}

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

{

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high- 1; j++)

{

if (arr[j] < pivot)

{

i++;

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

}

}

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

return (i + 1);

}

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

{

if (low < high)

{

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; 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;

}



**Program 8**

**Program for Radix Sort**

#include<stdio.h>

using namespace std;

int getMax(int arr[], int n)

{

int mx = arr[0];

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

if (arr[i] > mx)

mx = arr[i];

return mx;

}

void countSort(int arr[], int n, int exp)

{

int output[n];

int i, count[10] = {0};

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

count[ (arr[i]/exp)%10 ]++;

for (i = 1; i < 10; i++)

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

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

{

output[count[ (arr[i]/exp)%10 ] - 1] = arr[i];

count[ (arr[i]/exp)%10 ]--;

}

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

arr[i] = output[i];

}

void radixsort(int arr[], int n)

{

int m = getMax(arr, n);

for (int exp = 1; m/exp > 0; exp \*= 10)

countSort(arr, n, exp);

}

void print(int arr[], int n)

{

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

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

}

int main()

{

int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};

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

radixsort(arr, n);

print(arr, n);

return 0;

}



**Program 9**

**Program for Count Sort**

#include <stdio.h>

#include <string.h>

#define RANGE 255

void countSort(int n, int arr[])

{

int output[n];

int count[RANGE + 1], i;

memset(count, 0, sizeof(count));

for(i = 0; arr[i]; ++i)

++count[arr[i]];

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

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

for (i = 0; arr[i]; ++i)

{

output[count[arr[i]]-1] = arr[i];

--count[arr[i]];

}

for (i = 0; arr[i]; ++i)

arr[i] = output[i];

}

int main()

{

int arr[] = {9, 16, 20, 22, 27, 28, 36, 49};

int n = 8;

int i;

countSort(n,arr);

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

{

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

}

return 0;

}



**Program 10**

**Knapsack problem using Greedy Solution**

#include <stdio.h>

int n = 5;

int c[10] = {12, 1, 2, 1, 4};

int v[10] = {4, 2, 2, 1, 10};

int W = 15;

void simple\_fill() {

int cur\_w;

float tot\_v;

int i, maxi;

int used[10];

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

used[i] = 0;

cur\_w = W;

while (cur\_w > 0) {

maxi = -1;

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

if ((used[i] == 0) &&

((maxi == -1) || ((float)v[i]/c[i] > (float)v[maxi]/c[maxi])))

maxi = i;

used[maxi] = 1;

cur\_w -= c[maxi];

tot\_v += v[maxi];

if (cur\_w >= 0)

printf("Added object %d (%d$, %dKg) completely in the bag. Space left: %d.\n", maxi + 1, v[maxi], c[maxi], cur\_w);

else {

printf("Added %d%% (%d$, %dKg) of object %d in the bag.\n", (int)((1 + (float)cur\_w/c[maxi]) \* 100), v[maxi], c[maxi], maxi + 1);

tot\_v -= v[maxi];

tot\_v += (1 + (float)cur\_w/c[maxi]) \* v[maxi];

}

}

printf("Filled the bag with objects worth %.2f$.\n", tot\_v);

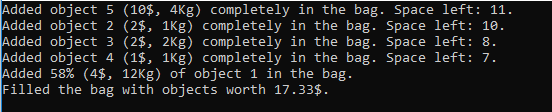
}

int main(int argc, char \*argv[]) {

simple\_fill();

return 0;

}



**Program 11**

**Travelling Salesman Problem**

#include<stdio.h>

int ary[10][10],completed[10],n,cost=0;

int least(int );

void takeInput()

{ int i,j;

printf("Enter the number of villages: ");

scanf("%d",&n);

printf("\nEnter the 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",&ary[i][j]);

completed[i]=0;

}

printf("\n\nThe cost list is:");

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

{

printf("\n");

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

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

}

}

void mincost(int city)

{ int i,ncity;

completed[city]=1;

printf("%d--->",city+1);

ncity=least(city);

if(ncity==999)

{

ncity=0;

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

cost+=ary[city][ncity];

return;

}

mincost(ncity);

}

int least(int c)

{ int i,nc=999;

int min=999,kmin;

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

{

if((ary[c][i]!=0)&&(completed[i]==0))

if(ary[c][i]+ary[i][c] < min)

{

min=ary[i][0]+ary[c][i];

kmin=ary[c][i];

nc=i;

}

}

if(min!=999)

cost+=kmin;

return nc;

}

int main()

{

takeInput();

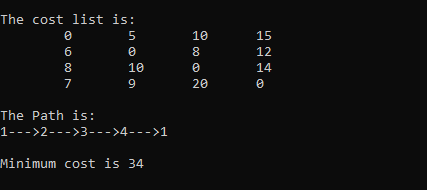
printf("\n\nThe Path is:\n");

mincost(0);

printf("\n\nMinimum cost is %d\n ",cost);

return 0;

}



**Program 12**

**Minimum spanning tree using Kruskal algorithm**

#include<stdio.h>

#define MAX 30

typedef struct edge

{

int u,v,w;

}edge;

typedef struct edgelist

{

edge data[MAX];

int n;

}edgelist;

edgelist elist;

int G[MAX][MAX],n;

edgelist spanlist;

void kruskal();

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

void union1(int belongs[],int c1,int c2);

void sort();

void print();

int main()

{

int i,j,total\_cost;

printf("\nEnter number 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]);

kruskal();

print();

return 0;

}

void kruskal()

{

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

elist.n=0;

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

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

{

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

{

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

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

elist.data[elist.n].w=G[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;

union1(belongs,cno1,cno2);

}

}

}

int find(int belongs[],int vertexno)

{

return(belongs[vertexno]);

}

void union1(int belongs[],int c1,int c2)

{

int i;

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

if(belongs[i]==c2)

belongs[i]=c1;

}

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;

}

}

void print()

{

int i,cost=0;

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

{

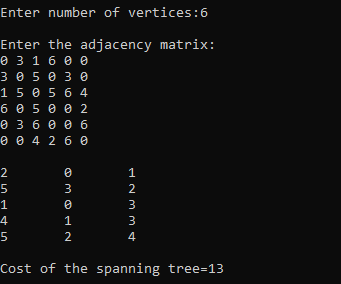
printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);

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

}

printf("\n\nCost of the spanning tree=%d",cost);

}



**Program 13**

**Implementation of N Queen problem using back tracking.**

#define N 4

#include <stdbool.h>

#include <stdio.h>

void printSolution(int board[N][N])

{

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

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

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

printf("\n");

}

}

bool isSafe(int board[N][N], int row, int col)

{

int i, j;

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

if (board[row][i])

return false;

for (i = row, j = col; i >= 0 && j >= 0; i--, j--)

if (board[i][j])

return false;

for (i = row, j = col; j >= 0 && i < N; i++, j--)

if (board[i][j])

return false;

return true;

}

bool solveNQUtil(int board[N][N], int col)

{

if (col >= N)

return true;

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

if (isSafe(board, i, col)) {

board[i][col] = 1;

if (solveNQUtil(board, col + 1))

return true;

board[i][col] = 0;

}

}

return false;

}

bool solveNQ()

{

int board[N][N] = { { 0, 0, 0, 0 },

{ 0, 0, 0, 0 },

{ 0, 0, 0, 0 },

{ 0, 0, 0, 0 } };

if (solveNQUtil(board, 0) == false) {

printf("Solution does not exist");

return false;

}

printSolution(board);

return true;

}

int main()

{

solveNQ();

return 0;

} 