**Bubble sort**

//Bubble sorting

#include <iostream>

using namespace std;

int main()

{

int i,n,temp=0,j;

cout<<"Enter the length";

cin>>n;

int arr[n];

cout<<"Enter the array";

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

cin>>arr[i];

}

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

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

if(arr[i]>arr[i+1]){

temp=arr[i];

arr[i]=arr[i+1];

arr[i+1]=temp;

}

}

}

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

cout<<arr[i]<<" ";

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Insertion SOrt**

Online C++ Compiler.

Code, Compile, Run and Debug C++ program online.

Write your code in this editor and press "Run" button to compile and execute it.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <bits/stdc++.h>

using namespace std;

// Function to sort an array using

// insertion sort

void insertionSort(int arr[], int n)

{

int i, key, j;

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

{

key = arr[i];

j = i - 1;

// Move elements of arr[0..i-1],

// that are greater than key, to one

// position ahead of their

// current position

while (j >= 0 && arr[j] > key)

{

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

j = j - 1;

}

arr[j + 1] = key;

}

}

// A utility function to print an array

// of size n

void printArray(int arr[], int n)

{

int i;

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

cout << arr[i] << " ";

cout << endl;

}

// Driver code

int main()

{

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

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

insertionSort(arr, N);

printArray(arr, N);

return 0;

}

Selection SOrt

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Online C++ Compiler.

Code, Compile, Run and Debug C++ program online.

Write your code in this editor and press "Run" button to compile and execute it.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// C++ program for implementation of

// selection sort

#include <bits/stdc++.h>

using namespace std;

//Swap function

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

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

// One by one move boundary of

// unsorted subarray

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

{

// Find the minimum element in

// unsorted array

min\_idx = i;

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

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

min\_idx = j;

// Swap the found minimum element

// with the first element

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

}

}

//Function to print an array

void printArray(int arr[], int size)

{

int i;

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

cout << arr[i] << " ";

cout << endl;

}

// Driver program to test above functions

int main()

{

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

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

selectionSort(arr, n);

cout << "Sorted array: \n";

printArray(arr, n);

return 0;

}

**Linear search**

//Linear Search

#include <iostream>

using namespace std;

int linearSearch(int n, int arr[], int key){

int i;

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

if(arr[i]==key){

return i;

}

}

return -1;

}

int main()

{

int n, i, key;

cout<<"Enter array's length";

cin>>n;

int arr[n];

cout<<"Enter the array";

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

cin>>arr[i];

}

cout<<"Enter the key";

cin>>key;

cout<<linearSearch(n,arr,key);

}

**Binary Search**

//Bineary Search

#include <iostream>

using namespace std;

int binarySearch(int n, int arr[], int key){

int s=0, e=n-1,mid;

while(s<=e){

mid=(s+e)/2;

if(key==arr[mid]){

return mid;

}

else if(key<arr[mid]){

e=mid-1;

}

else{

s=mid+1;

}

}

}

int main()

{

int n,key;

cout<<"Enter length of array";

cin>>n;

int arr[n];

cout<<"Enter the array";

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

cin>>arr[i];

}

cout<<"Enter the key";

cin>>key;

cout<<binarySearch(n,arr,key);

}

**Binary search Recursion**

**#include <bits/stdc++.h>**

**using namespace std;**

**// A recursive binary search function. It returns**

**// location of x in given array arr[l..r] is present,**

**// otherwise -1**

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

**{**

**if (r >= l) {**

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

**// If the element is present at the middle**

**// itself**

**if (arr[mid] == x)**

**return mid;**

**// If element is smaller than mid, then**

**// it can only be present in left subarray**

**if (arr[mid] > x)**

**return binarySearch(arr, l, mid - 1, x);**

**// Else the element can only be present**

**// in right subarray**

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

**}**

**// We reach here when element is not**

**// present in array**

**return -1;**

**}**

**int main(void)**

**{**

**int arr[] = { 2, 3, 4, 10, 40 };**

**int x = 10;**

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

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

**(result == -1)**

**? cout << "Element is not present in array"**

**: cout << "Element is present at index " << result;**

**return 0;**

**}**

**Quick sort**

#include <bits/stdc++.h>

using namespace std;

// A utility function to swap two elements

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

{

int t = \*a;

\*a = \*b;

\*b = t;

}

/\* This function takes last element as pivot, places

the pivot element at its correct position in sorted

array, and places all smaller (smaller than pivot)

to left of pivot and all greater elements to right

of pivot \*/

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

{

int pivot = arr[high]; // pivot

int i = (low - 1); // Index of smaller element and indicates the right position of pivot found so far

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

{

// If current element is smaller than the pivot

if (arr[j] < pivot)

{

i++; // increment index of smaller element

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

}

}

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

return (i + 1);

}

/\* The main function that implements QuickSort

arr[] --> Array to be sorted,

low --> Starting index,

high --> Ending index \*/

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

{

if (low < high)

{

/\* pi is partitioning index, arr[p] is now

at right place \*/

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

// Separately sort elements before

// partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

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

cout << arr[i] << " ";

cout << endl;

}

// Driver Code

int main()

{

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

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

quickSort(arr, 0, n - 1);

cout << "Sorted array: \n";

printArray(arr, n);

return 0;

}

Merge Sort

// Merge sort in C++

#include <iostream>

using namespace std;

// Merge two subarrays L and M into arr

void merge(int arr[], int p, int q, int r) {

// Create L ← A[p..q] and M ← A[q+1..r]

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], M[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[p + i];

for (int j = 0; j < n2; j++)

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

// Maintain current index of sub-arrays and main array

int i, j, k;

i = 0;

j = 0;

k = p;

// Until we reach either end of either L or M, pick larger among

// elements L and M and place them in the correct position at A[p..r]

while (i < n1 && j < n2) {

if (L[i] <= M[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = M[j];

j++;

}

k++;

}

// When we run out of elements in either L or M,

// pick up the remaining elements and put in A[p..r]

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = M[j];

j++;

k++;

}

}

// Divide the array into two subarrays, sort them and merge them

void mergeSort(int arr[], int l, int r) {

if (l < r) {

// m is the point where the array is divided into two subarrays

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

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

// Merge the sorted subarrays

merge(arr, l, m, r);

}

}

// Print the array

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

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

cout << arr[i] << " ";

cout << endl;

}

// Driver program

int main() {

int arr[] = {6, 5, 12, 10, 9, 1};

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

mergeSort(arr, 0, size - 1);

cout << "Sorted array: \n";

printArray(arr, size);

return 0;

}

**Heap SOrt**

// C++ program for implementation of Heap Sort

#include <iostream>

using namespace std;

// To heapify a subtree rooted with node i which is

// an index in arr[]. n is size of heap

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

{

int largest = i; // Initialize largest as root

int l = 2 \* i + 1; // left = 2\*i + 1

int r = 2 \* i + 2; // right = 2\*i + 2

// If left child is larger than root

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

largest = l;

// If right child is larger than largest so far

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

largest = r;

// If largest is not root

if (largest != i) {

swap(arr[i], arr[largest]);

// Recursively heapify the affected sub-tree

heapify(arr, n, largest);

}

}

// main function to do heap sort

void heapSort(int arr[], int n)

{

// Build heap (rearrange array)

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

heapify(arr, n, i);

// One by one extract an element from heap

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

// Move current root to end

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

// call max heapify on the reduced heap

heapify(arr, i, 0);

}

}

/\* A utility function to print array of size n \*/

void printArray(int arr[], int n)

{

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

cout << arr[i] << " ";

cout << "\n";

}

// Driver program

int main()

{

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

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

heapSort(arr, n);

cout << "Sorted array is \n";

printArray(arr, n);

}

**Radiax SOrt**

**#include <iostream>**

**using namespace std;**

**// A utility function to get maximum value in arr[]**

**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;**

**}**

**// A function to do counting sort of arr[] according to**

**// the digit represented by exp.**

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

**{**

**int output[n]; // output array**

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

**// Store count of occurrences in count[]**

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

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

**// Change count[i] so that count[i] now contains actual**

**// position of this digit in output[]**

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

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

**// Build the output array**

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

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

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

**}**

**// Copy the output array to arr[], so that arr[] now**

**// contains sorted numbers according to current digit**

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

**arr[i] = output[i];**

**}**

**// The main function to that sorts arr[] of size n using**

**// Radix Sort**

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

**{**

**// Find the maximum number to know number of digits**

**int m = getMax(arr, n);**

**// Do counting sort for every digit. Note that instead**

**// of passing digit number, exp is passed. exp is 10^i**

**// where i is current digit number**

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

**countSort(arr, n, exp);**

**}**

**// A utility function to print an array**

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

**{**

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

**cout << arr[i] << " ";**

**}**

**// Driver Code**

**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;**

**}**

**Optimal Merge**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

int optimalMerge(int n, int files[]){

priority\_queue<int, vector<int>, greater<int>> pq;

int i;

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

pq.push(files[i]);

}

int count =0;

while(pq.size()>1){

int first\_smallest= pq.top();

pq.pop();

int second\_smallest = pq.top();

pq.pop();

int temp = first\_smallest + second\_smallest;

count+=temp;

pq.push(temp);

}

return count;

}

int main()

{

int n=6;

int files[6]={1,5,3,8,9,4};

cout<<optimalMerge(n,files);

return 0;

}

**Knapsack**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

struct item{

int value,weight;

};

bool compare(item a, item b){

double r1= (double)a.value/a.weight;

double r2= (double)b.value/b.weight;

return (r1>r2);

}

void knapsack(item arr[], int C, int size){

int i;

sort(arr,arr+size,compare);

int curr\_weight=0;

double final\_value=0.0;

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

if((curr\_weight+arr[i].weight<=C)){

curr\_weight+=arr[i].weight;

final\_value+=arr[i].value;

}

else{

int remain= C- curr\_weight;

final\_value+=arr[i].value\*((double)remain/arr[i].weight);

}

}

cout<<"final value:"<<final\_value;

}

int main()

{

int C=60;

item arr[]{

{100,10},

{120,40},

{80,20},

{88,30}

};

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

cout<<"The array:";

knapsack(arr,C,size);

return 0;

}

**Dijkstra**

#include<iostream>

#include<climits>

using namespace std;

int miniDist(int distance[], bool Tset[]) // finding minimum distance

{

int minimum=INT\_MAX,ind;

for(int k=0;k<6;k++)

{

if(Tset[k]==false && distance[k]<=minimum)

{

minimum=distance[k];

ind=k;

}

}

return ind;

}

void dijkstraAlgo(int graph[6][6],int src) // adjacency matrix

{

int distance[6]; // // array to calculate the minimum distance for each node

bool Tset[6];// boolean array to mark visited and unvisited for each node

for(int k = 0; k<6; k++)

{

distance[k] = INT\_MAX;

Tset[k] = false;

}

distance[src] = 0; // Source vertex distance is set 0

for(int k = 0; k<6; k++)

{

int m=miniDist(distance,Tset);

Tset[m]=true;

for(int k = 0; k<6; k++)

{

// updating the distance of neighbouring vertex

if(!Tset[k] && graph[m][k] && distance[m]!=INT\_MAX && distance[m]+graph[m][k]<distance[k])

distance[k]=distance[m]+graph[m][k];

}

}

cout<<"Vertex\t\tDistance from source vertex"<<endl;

for(int k = 0; k<6; k++)

{

char str=65+k;

cout<<str<<"\t\t\t"<<distance[k]<<endl;

}

}

int main()

{

int graph[6][6]={

{0,1,3,3,0,2},

{1,2,2,3,0,0},

{3,2,3,2,6,0},

{3,3,2,5,0,2},

{0,0,6,0,4,5},

{2,0,0,2,5,0}

};

dijkstraAlgo(graph,0);

return 0;

}

**Job sequencing**

**#include <algorithm>**

**#include <iostream>**

**using namespace std;**

**// A structure to represent a job**

**struct Job {**

**char id; // Job Id**

**int dead; // Deadline of job**

**int profit; // Profit if job is over before or on**

**// deadline**

**};**

**// This function is used for sorting all jobs according to**

**// profit**

**bool comparison(Job a, Job b)**

**{**

**return (a.profit > b.profit);**

**}**

**// Returns minimum number of platforms required**

**void printJobScheduling(Job arr[], int n)**

**{**

**// Sort all jobs according to decreasing order of profit**

**sort(arr, arr + n, comparison);**

**int result[n]; // To store result (Sequence of jobs)**

**bool slot[n]; // To keep track of free time slots**

**// Initialize all slots to be free**

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

**slot[i] = false;**

**// Iterate through all given jobs**

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

**// Find a free slot for this job (Note that we start**

**// from the last possible slot)**

**for (int j = min(n, arr[i].dead) - 1; j >= 0; j--) {**

**// Free slot found**

**if (slot[j] == false) {**

**result[j] = i; // Add this job to result**

**slot[j] = true; // Make this slot occupied**

**break;**

**}**

**}**

**}**

**// Print the result**

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

**if (slot[i])**

**cout << arr[result[i]].id << " ";**

**}**

**// Driver code**

**int main()**

**{**

**Job arr[] = { { 'a', 2, 100 },**

**{ 'b', 1, 19 },**

**{ 'c', 2, 27 },**

**{ 'd', 1, 25 },**

**{ 'e', 3, 15 } };**

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

**cout << "Following is maximum profit sequence of jobs "**

**"\n";**

**// Function call**

**printJobScheduling(arr, n);**

**return 0;**

**}**

**// This code is contributed by Aditya Kumar (adityakumar129)**

**Kruskal**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

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

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

int find(int);

int uni(int, int);

void main()

{

printf("Kruskal's algorithm in C\n");

printf("========================\n");

printf("Enter the no. of vertices:\n");

scanf("%d", &n);

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

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

{

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

{

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

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

cost[i][j] = 999;

}

}

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

while (ne < n)

{

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

{

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

{

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

{

min = cost[i][j];

a = u = i;

b = v = j;

}

}

}

u = find(u);

v = find(v);

if (uni(u, v))

{

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

mincost += min;

}

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

}

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

getch();

}

int find(int i)

{

while (parent[i])

i = parent[i];

return i;

}

int uni(int i, int j)

{

if (i != j)

{

parent[j] = i;

return 1;

}

return 0;

}

**Prims**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

int main()

{

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

printf("Enter vertices");

scanf("%d",&n);

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

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

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

if(cost[i][j]==0){

cost[i][j]=999;

}

}

}

visited[1]=1;

while(ne<n){

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

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

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

if(visited[i]!=0){

min=cost[i][j];

a=u=i;

a=v=j;

}

}

}

}

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

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

mincost+=min;

visited[b]=1;

}

}

return 0;

}