**Fibonacci Series:**

The Fibonacci sequence is the series of numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

The next number is found by adding up the two numbers before it.

* The 2 is found by adding the two numbers before it (1+1)
* The 3 is found by adding the two numbers before it (1+2),
* And the 5 is (2+3),
* and so on!

**Fibonacci Sequence Using Recursion.**

#include<stdio.h>

void printfibonacci(int);

int main()

{

int k,n;

printf("enter the range of fibonacci Series");

scanf("%d",&n);

printf("fibonacci series:");

printf("%d%d",0,1);

printfibonacci(n);

return 0;

}

void printfibonacci(int n)

{

static long int first=0,second=1,sum;

if(n>0)

{

sum=first+second;

first=second;

second=sum;

printf("%d",sum);

printfibonacci(n-1);

}

}

**Fibonacci Series using Non-Recursion**

#include<stdio.h>

int main()

{

int n,first=0,second=1,next,c;

clrscr();

printf("enter the range of the fibonacci series:");

scanf("%d",&n);

printf("first %d terms of fibonacci series are \n",n);

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

{

if(c<=1)

next=c;

else

{

next=first+second;

first=second;

second=next;

}

printf("%d\n",next);

}

getch();

return 0;

}

**Factorial of a number**

The factorial function (symbol: !) means to multiply a series of descending natural numbers. Examples:

* 4! = 4 × 3 × 2 × 1 = 24
* 7! = 7 × 6 × 5 × 4 × 3 × 2 × 1 = 5040
* 1! = 1

**Factorial of an integer using Recursive and non-recursive**

#include<stdio.h>

#include<conio.h>

void main()

{

int n,a,b;

clrscr();

printf("enter any number\n");

scanf("%d",&n);

a=refactorial(n);

printf("the factorial of a given number using recursion is %d\n",a);

b=nonrefactorial(n);

printf("the factorial of a given number using nonrecursion is %d",b);

getch();

}

int refactorial(int x)

{

int f;

if(x==0)

{

return(1);

}

else

{

f=x\*refactorial(x-1);

return(f);

}

}

int nonrefactorial(int x)

{

int i,f=1;

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

{

f=f\*i;

}

return(f);

}

**GCD of Two Integers**

**Greatest Common Factor of 12 and 16**

* **Find all the Factors of each number,**
* **Circle the Common factors,**
* **Choose the Greatest of those**

**GCD of two Integers**

#include <stdio.h>

#include <conio.h>

void main()

{

int a, b, c, d;

clrscr();

printf("Enter two numbers a, b\n");

scanf("%d%d", &a, &b);

c = recgcd(a, b);

printf("The gcd of two numbers using recursion is %d\n", c);

d = nonrecgcd(a, b);

printf("The gcd of two numbers using nonrecursion is %d", d);

getch();

}

int recgcd(int x, int y)

{

if(y == 0)

{

return(x);

}

else

{

return(recgcd(y, x % y));

}

}

int nonrecgcd(int x, int y)

{

int z;

while(x % y != 0)

{

z = x % y;

x = y;

y = z;

}

return(y);

}

Enter two numbers a, b

3 6

The gcd of two numbers using recursion is 3

The gcd of two numbers using nonrecursion is 3

**Selection sort:**

Selection sort algorithm starts by comparing first two elements of an array and swapping if necessary, i.e., if you want to sort the elements of array in ascending order and if the first element is greater than second then, you need to swap the elements but, if the first element is smaller than second, leave the elements as it is. Then, again first element and third element are compared and swapped if necessary. This process goes on until first and last element of an array is compared. This completes the first step of selection sort.

If there are n elements to be sorted then, the process mentioned above should be repeated n-1 times to get required result. But, for better performance, in second step, comparison starts from second element because after first step, the required number is automatically placed at the first (i.e, In case of sorting in ascending order, smallest element will be at first and in case of sorting in descending order, largest element will be at first.). Similarly, in third step, comparison starts from third element and so on.



**Program for sorting the given list using Selection sort**

#include <stdio.h>

int main()

{

int data[100],i,n,steps,temp;

printf("Enter the number of elements to be sorted: ");

scanf("%d",&n);

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

{

printf("%d. Enter element: ",i+1);

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

}

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

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

{

if(data[steps]>data[i])

/\* To sort in descending order, change > to <. \*/

{

temp=data[steps];

data[steps]=data[i];

data[i]=temp;

}

}

printf("In ascending order: ");

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

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

return 0;

}

**Output**

Enter the number of elements to be sorted: 5

1. Enter element: 12

2. Enter element: 1

3. Enter element: 23

4. Enter element: 2

5. Enter element: 0

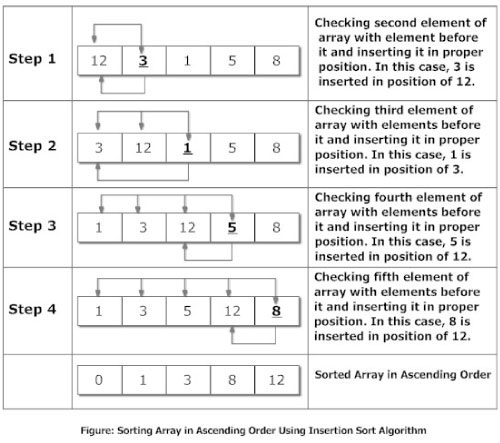
In ascending order: 0 1 2 12 23

**Insertion Sort**

Suppose, you want to sort elements in ascending as in above figure. Then,

1. Step 1: The second element of an array is compared with the elements that appears before it (only first element in this case). If the second element is smaller than first element, second element is inserted in the position of first element. After first step, first two elements of an array will be sorted.
2. Step 2: The third element of an array is compared with the elements that appears before it (first and second element). If third element is smaller than first element, it is inserted in the position of first element. If third element is larger than first element but, smaller than second element, it is inserted in the position of second element. If third element is larger than both the elements, it is kept in the position as it is. After second step, first three elements of an array will be sorted.
3. Step 3: Similary, the fourth element of an array is compared with the elements that appears before it (first, second and third element) and the same procedure is applied and that element is inserted in the proper position. After third step, first four elements of an array will be sorted.

If there are n elements to be sorted. Then, this procedure is repeated n-1 times to get sorted list of array.



**Program for sorting the given list using insertion sort.**

#include<stdio.h>

int main()

{

int data[100],n,temp,i,j;

printf("Enter number of terms(should be less than 100): ");

scanf("%d",&n);

printf("Enter elements: ");

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

{

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

}

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

{

temp = data[i];

j=i-1;

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

/\*To sort elements in descending order, change temp<data[j] to temp>data[j] in above line.\*/

{

data[j+1] = data[j];

--j;

}

data[j+1]=temp;

}

printf("In ascending order: ");

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

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

return 0;

}

**Output**

Enter number of terms(should be less than 100): 5

Enter elements: 12

1

2

5

3

In ascending order: 1 2 3 5 12

**Merge Sort**

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being Ο(n log n), it is one of the most respected algorithms.

Merge sort first divides the array into equal halves and then combines them in a sorted manner.

**How Merge Sort Works?**

To understand merge sort, we take an unsorted array as the following −

Unsorted Array

We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.

Merge Sort Division

This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

Merge Sort Division

We further divide these arrays and we achieve atomic value which can no more be divided.

Merge Sort Division

Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.

We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.

Merge Sort Combine

In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.

Merge Sort Combine

After the final merging, the list should look like this −

Merge Sort

**Program for sorting the given list using Merge sort.**

#include<stdio.h>

#include<conio.h>

int a[50];

void merge(int,int,int);

void merge\_sort(int low,int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

merge\_sort(low,mid);

merge\_sort(mid+1,high);

merge(low,mid,high);

}

}

void merge(int low,int mid,int high)

{

int h,i,j,b[50],k;

h=low;

i=low;

j=mid+1;

while((h<=mid)&&(j<=high))

{

if(a[h]<=a[j])

{

b[i]=a[h];

h++;

}

else

{

b[i]=a[j];

j++;

}

i++;

}

if(h>mid)

{

for(k=j;k<=high;k++)

{

b[i]=a[k];

i++;

}

}

else

{

for(k=h;k<=mid;k++)

{

b[i]=a[k];

i++;

}

}

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

a[k]=b[k];

}

void main()

{

int num,i;

clrscr();

printf("\t\t mergesort\n");

printf("\n Enter the total numbers:");

scanf("%d",&num);

printf("\n Enter %d numbers:\n",num);

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

{

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

}

merge\_sort(1,num);

printf("\n SORTED ORDER:\n");

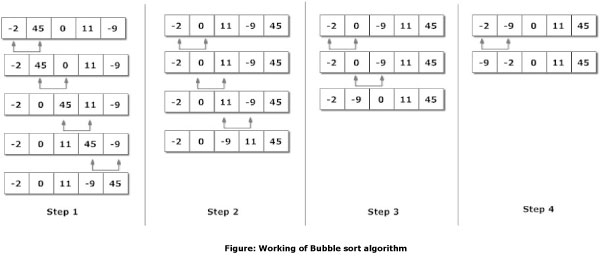
for(i=1;i<=num;i++)

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

**}**

**Bubble Sort**

Bubble sort is based on the idea of repeatedly comparing pairs of adjacent elements and then swapping their positions if they exist in the wrong order.



**Program for sorting the given list using Bubble sort.**

#include<stdio.h>

#include<conio.h>

void bubblesort(int \*a,int n);

int main()

{

int arr[5],i,n;

clrscr();

printf("enter no.of elements to be sorted\n");

scanf("%d",&n);

printf("enter %d integers\n",n);

for(i=0;i<5;i++)

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

bubblesort(arr,5);

printf("sorted array is\n");

for(i=0;i<5;i++)

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

getch();

return 0;

}

void bubblesort(int \*a,int n)

{

int k,j,temp;

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

{

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

{

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

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

}

# Heap Sort

Heaps can be used in sorting an array. In max-heaps, maximum element will always be at the root. Heap Sort uses this property of heap to sort the array.

Consider an array ArrArr which is to be sorted using Heap Sort.

* Initially build a max heap of elements in ArrArr.
* The root element, that is Arr[1]Arr[1], will contain maximum element of ArrArr. After that, swap this element with the last element of ArrArr and heapify the max heap excluding the last element which is already in its correct position and then decrease the length of heap by one.
* Repeat the step 2, until all the elements are in their correct position.

**Program for sorting the given list using Heap sort**

#include<stdio.h>

void heapsort(int[],int);

void heapify(int[],int);

void adjust(int[],int);

void main()

{

int n,i,a[50];

system("clear");

printf("\nEnter the limit:");

scanf("%d",&n);

printf("\nEnter the elements:");

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

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

heapsort(a,n);

printf("\nThe Sorted Elements Are:\n");

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

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

printf("\n");

}

void heapsort(int a[],int n) {

int i,t;

heapify(a,n);

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

t = a[0];

a[0] = a[i];

a[i] = t;

adjust(a,i);

}

}

void heapify(int a[],int n) {

int k,i,j,item;

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

item = a[k];

i = k;

j = (i-1)/2;

while((i>0)&&(item>a[j])) {

a[i] = a[j];

i = j;

j = (i-1)/2;

}

a[i] = item;

}

}

void adjust(int a[],int n) {

int i,j,item;

j = 0;

item = a[j];

i = 2\*j+1;

while(i<=n-1) {

if(i+1 <= n-1)

if(a[i] <a[i+1])

i++;

if(item<a[i]) {

a[j] = a[i];

j = i;

i = 2\*j+1;

} else

break;

}

a[j] = item;

}

**Binary Search**

Binary search is a fast search algorithm with run-time complexity of Ο(log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.

Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues on the sub-array as well until the size of the subarray reduces to zero.

**Program to find the given number in a list using Binary Search.**

#include<stdio.h>

int main()

{

int i,first,last,middle,n,search,array[100];

printf("enter no.of elements\n");

scanf("%d",&n);

printf("enter %d integers\n",n);

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

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

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;

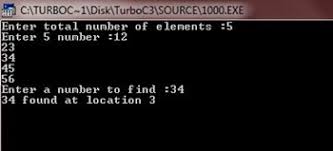
middle=(first+last)/2;

}

if(first>last)

printf("not found %d is not present");

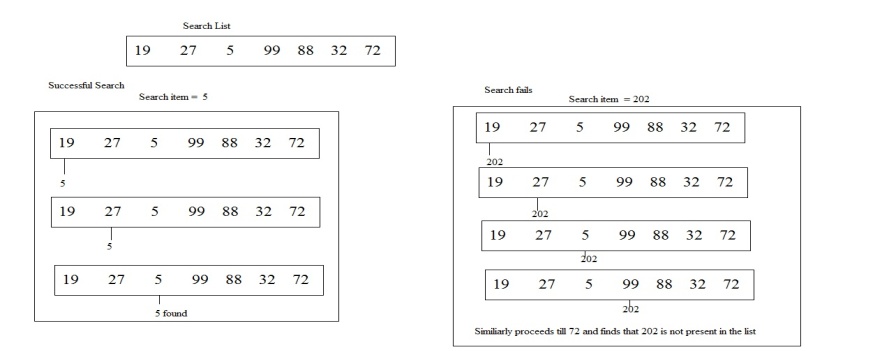
return 0;

}

**Sequential Search**

A simple approach is to do **linear search**, i.e

* Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
* If x matches with an element, return the index.
* If x doesn’t match with any of elements, return -1.



Program to find the given numbers in a list using Sequential Search.

#include<stdio.h>

int main()

{

int array[100],search,c,n;

clrscr();

printf("enter the no.of elements in array\n");

scanf("%d",&n);

printf("enter %d integers \n",n);

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

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

printf("enter the number to serach\n");

scanf("%d",&search);

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

{

if(array[c]==search)

{

printf("%d is present at location %d\n",search,c+1);

break;

}

}

if(c==n)

printf("%d is not present in array\n",search);

getch();

return 0;

}

**Program to find the Product of two Matrices**

#include<stdio.h>

int main()

{

int a[3][3],b[3][3],c[3][3],i,j,k;

clrscr();

printf("enter the elements of Matrix A");

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

{

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

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

}

printf("enter the elements of Matrix B");

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

{

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

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

}

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

{

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

{

c[i][j]=0;

for(k=0;k<=2;k++)

{

c[i][j]=c[i][j]+a[i][k]\*b[k][j];

}

}

}

printf("matrix mul=",c[i][j]);

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

{

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

{

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

}

}

getch();

return 0;

}

Knapsack Problem.

1. Given a set of items, each with a **weight and a value**.
2. Determine the **number of each item** to include in a collection so that the total weight is less than a given limit and the total value is as large as possible.
3. It derives its name from the problem faced by someone who is constrained by a **fixed-size knapsack** and must fill it with the most useful items.

Write a program to solve knapsack problem

# include<stdio.h>

#include<conio.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;

clrscr();

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);

getch();

return(0);

}

Output:

Enter the no. of objects:- 7

Enter the wts and profits of each object:-

2 10

3 5

5 15

7 7

1 6

4 18

1 3

Enter the capacity of knapsack:- 15

The result vector is:- 1.000000 1.000000 1.000000 1.000000

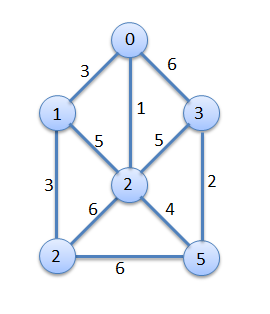
1.000000 0.666667 0.000000

Maximum profit is:- 55.333332

# Kruskal's Algorithm

Kruskal's algorithm is a greedy algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph.   
It finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.  
This algorithm is directly based on the MST( minimum spanning tree) property.

Example:

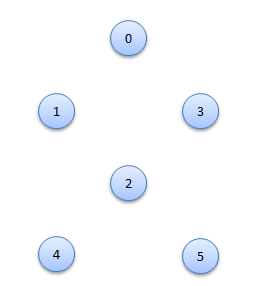


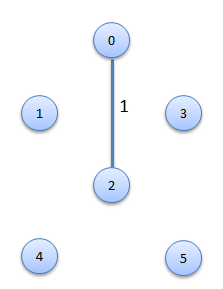
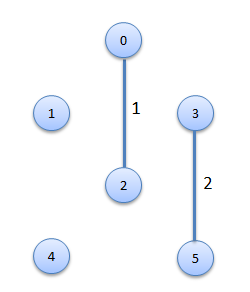
#### Procedure for finding Minimum Spanning Tree

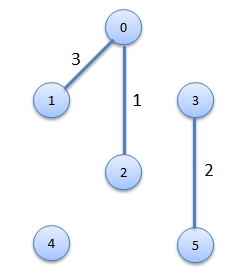
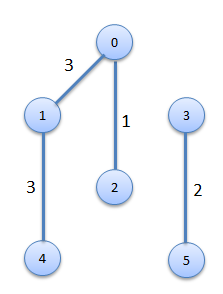
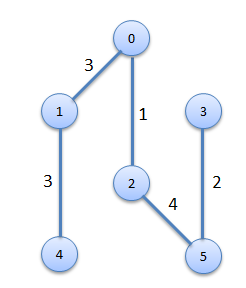
**Step1.** Edges are sorted in ascending order by weight.

|  |  |  |
| --- | --- | --- |
| **Edge No.** | **Vertex Pair** | **Edge Weight** |
| E1 | (0,2) | 1 |
| E2 | (3,5) | 2 |
| E3 | (0,1) | 3 |
| E4 | (1,4) | 3 |
| E5 | (2,5) | 4 |
| E6 | (1,2) | 5 |
| E7 | (2,3) | 5 |
| E8 | (0,3) | 6 |
| E9 | (2,4) | 6 |
| E10 | (4,5) | 6 |

**Step2.** Edges are added in sequence.

Graph             

Add Edge E1  Add Edge E2

Add Edge E3 Add Edge E4 Add Edge E5

Program to find the Minimum Spanning Tree for a weighted Graph Using Kruskals Algorithm

#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()

{

clrscr();

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);

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;

}

Quicksort

Quicksort is a divide and conquer algorithm. The steps are: 1) Pick an element from the array, this element is called as pivot element. 2) Divide the unsorted array of elements in two arrays with values less than the pivot come in the first sub array, while all elements with values greater than the pivot come in the second sub-array (equal values can go either way). This step is called the partition operation. 3) Recursively repeat the step 2(until the sub-arrays are sorted) to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

Program for sorting the given list using Quick sort

#include<stdio.h>

#include<conio.h>

int arr[40];

void quicksort(int a[],int p,int r);

int partition(int a[],int p,int r);

void exchange(int i,int j);

void quicksort(int a[],int p,int r){

int q;

if(p<r){

q=partition(a,p,r);

quicksort(a,p,q-1);

quicksort(a,q+1,r);

}

}

int partition(int a[],int p,int r){

int x,j,i;

x=a[r];

i=p-1;

for(j=p;j<=(r-1);j++)

if(a[j]<x){

i=i+1;

exchange(i,j);

}

exchange(i+1,r);

return(i+1);

}

void exchange(int i,int j){

int temp;

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

int main(){

int n,i;

printf("\nEnter no of elements needed :" );

scanf("%d",&n);

printf("\nEnter elements : ");

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

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

quicksort(arr,1,n);

printf("\nSorted Array is : ");

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

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

getch();

return(0);

}