```
//muestra los 10 primeros terminos
//de la sucesion de Fibonacci
//usando una funcion recursiva
#include<iostream>
using namespace std;

//devuelve el termino i esimo de la
//sucesion de fibonacci
int fibonacci(int i)
{
  //casos base, los dos primeros
  //terminos son 1 por definicion
  if (i==0||i==1)
    return 1;
  //llamada recursiva
  //los demas son la suma de los dos anteriores
  else
    return fibonacci(i-1)+fibonacci(i-2);
}

int main()
{
  for (int i=0;i<10;i++)
    cout<<fibonacci(i)<<endl;
  cout<<c;
}
```

```
//suma de los elementos de un vector
//usando recursividad, dos implementaciones
//distintas
#include<iostream>
using namespace std;
#include<vector>

//una version recursiva no muy eficiente
int suma1(const vector<int> &a)
{
  //caso base, vector de tamaÃ±o 1
  //la suma es el unico elemento
  if (a.size()==1)
    return a[0];
  else
    //en caso contrario la suma es el primer
    //elemento mas los restantes
    return a[0]+suma1(vector<int>(a.begin()+1,a.end()));
}

//una version recursiva mas eficiente pero menos elegante
//necesita un parametro adicional para especificar
//desde donde se va a sumar
int suma2(const vector<int> &a,int i)
{
  //caso base, estamos al final del vector
  if (i==a.size()-1)
    //la suma hasta el final es el ultimo elemento
    return a[a.size()-1];
  else
    //en caso contrario la suma hasta el final es
    //el elemento actual mas la suma del resto hasta
    //el final
    return a[i]+suma2(a,i+1);
}

int main()
{
  vector<int> x(5);
  for (int i=0;i<x.size();i++)
    cin>>x[i];
  cout<<suma1(x)<<' '<<suma2(x,0);
}
```

```
//paridad de un vector de booleanos
//es true si numero par de bool
//false en caso contrario
//dos implementaciones recursivas distintas
#include<iostream>
using namespace std;
#include<vector>

//version recursiva
bool es_par2(const vector<bool>&a)
{
  //caso base, un solo elemento
  if (a.size()==1)
    if (a[0])
      return false;
    else
      return true;
  //llamadas recursivas
  else
    //si el primer elemento es true
    if (a[0])
      //esto no es eficiente, se crea un vector
      //con un elemento  menos cada vez
      //si el resto del vector es par
      if(es_par2(vector<bool>(a.begin()+1,a.end())))
        return false;
  //si el resto del vector es impar
      else
        return true;
  //si el primer elemento es false
    else
      //si el resto del vector es par
      if(es_par2(vector<bool>(a.begin()+1,a.end())))
        return true;
  //si el resto del vector es impar
      else
        return false;
}
```

```
//otra version recursiva. Adicionalmente, recibe un entero
//que especifica desde donde se comienza a procesar el vector
//mas eficiente, menos elegante
bool es_par3(const vector<bool>&a, int i)
{
  //caso base, estamos al final del vector
  if (i==a.size()-1)
    if (a[0])
      return false;
    else
      return true;
  //llamadas recursivas
  else
    //si el primer elemento es true
    if (a[i])
      //si el resto del vector es par
      if(es_par3(a,i+1))
        return false;
  //si el resto del vector es impar
      else
        return true;
  //si el primer elemento es false
    else
      //si el resto del vector es par
      if(es_par3(a,i+1))
        return true;
  //si el resto del vector es impar
      else
        return false;
}

int main()
{
  vector<bool> x(4);
  x[0]=true; x[1]=false; x[2]=false; x[3]=true;
  if (es_par2(x))
    cout<<"par"<<endl;
  else
    cout<<"impar"<<endl;
  if (es_par3(x,0))
    cout<<"par"<<endl;
  else
    cout<<"impar"<<endl;
  vector<bool> y(4);
  y[0]=true; y[1]=false; y[2]=true; y[3]=true;
  if (es_par2(y))
    cout<<"par"<<endl;
  else
    cout<<"impar"<<endl;
  if (es_par3(y,0))
    cout<<"par"<<endl;
  else
    cout<<"impar"<<endl;
}
```

```
//ordenacion de un vector de menor a mayor mediante
//el metodo merge-sort.
#include<iostream>
#include<vector>
using namespace std;

void mezcla(const vector<int>&x,const vector<int>&y,vector<int>&sal)
{
  int i_x,i_y,i_sal;
  i_x=0;
  i_y=0;
  i_sal=0;
  sal=vector<int>(x.size()+y.size());
  while(i_x<x.size()||i_y<y.size())
    {
      if (i_x<x.size()&&i_y<y.size())
        {
          if (x[i_x]<y[i_y])
            {
              sal[i_sal]=x[i_x];
              i_sal++;
              i_x++;
            }
          else
            {
              sal[i_sal]=y[i_y];
              i_sal++;
              i_y++;
            }
        }
      else
        {
          if (i_x<x.size())
            {
              sal[i_sal]=x[i_x];
              i_x++;
              i_sal++;
            }
          if (i_y<y.size())
            {
              sal[i_sal]=y[i_y];
              i_y++;
              i_sal++;
            }
        }
    }
}

//recibe un vector de entrada, los indices entre los que se va a ordenar
//y devuelve el vector ordenado en s
void ordena_r(const vector<int> &e, int ini, int fin, vector<int> &s)
{
  int mitad;
  //primer caso base, si hay un solo elemento
  if (ini==fin)
    {
      s=vector<int>(1);
      s[0]=e[ini];
    }
  else
    //segundo caso base, si hay dos elementos
    if (1==(fin-ini))
      {
        s=vector<int>(2);
        //si estan ordenados los guardo en el mismo orden
        if (e[ini]<=e[fin])
          {
            s[0]=e[ini];
            s[1]=e[fin];
          }
        //si no estan ordenados los guardo al reves
        else
          {
            s[0]=e[fin];
            s[1]=e[ini];
          }
      }
  //lamadas recursivas
    else
      {
        //calculo el punto medio de lo que estoy ordenando
        mitad=ini+(fin-ini)/2;
        //declaro los vectores que contendran las mitades ordenadas
        vector<int> mitad1(mitad-ini+1),mitad2(fin-mitad);
        //ordeno el vector entre ini y mitad, devuelvo el resultado en
        //mitad1
        ordena_r(e,ini,mitad,mitad1);
        //ordeno el vector entre mitad+1 y fin, devuelvo el resultado en
        //mitad2
        ordena_r(e,mitad+1,fin,mitad2);
        //mezclo las mitades ordenadas
        mezcla(mitad1,mitad2,s);
      }
}

void ordena(vector<int> &a)
{
  vector<int> b;
  ordena_r(a,0,a.size()-1,b);
  a=b;
}

istream & operator>>(istream &is, vector<int> &a)
{
  for (int i=0;i<a.size();i++)
    is>>a[i];
  return is;
}

ostream & operator<<(ostream &os, const vector<int> &a)
{
  for (int i=0;i<a.size();i++)
    os<<a[i]<<' ';
  return os;
}

int main()
{
  vector<int> x(5);
  cin>>x;
  ordena(x);
  cout<<x<<endl;
}
```