Problema 1

Se citește de la tastatură un vector cu **n** ($1 \leq n \leq 100$) elemente numere întregi. Să se calculeze suma lor.

```
#include <iostream>
using namespace std;

long long int suma(int v[], int st, int dr)
{
   //problema elementara
   if(st==dr)
      return v[st];

   //se imparte problema in subprobleme
   int mij=(st+dr)/2;
   long long int s1=suma(v, st, mij); //rezolvam prima subproblema
   long long int s2=suma(v, mij+1, dr); //rezolvam a doua subproblema

   //se combina rezultatele
   return s1+s2;
}

int main()
{
   int v[100], n;
   cin>>n;
   for(int i=0; i<n; i++)
      cin>>v[i];
   cout<<suma(v, 0, n-1);
   return 0;
}
```

Problema 2

Se citește de la tastatură un vector cu **n** ($1 \leq n \leq 100$) elemente numere întregi. Să se determine cel mai mare element.

```
#include <iostream>
using namespace std;

int maxim(int v[], int st, int dr)
{
   //problema elementara
   if(st==dr)
      return v[st];

   //se imparte problema in subprobleme
   int mij=(st+dr)/2;
   int max1=maxim(v, st, mij); //rezolvam prima subproblema
   int max2=maxim(v, mij+1, dr); //rezolvam a doua subproblema

   //se combina rezultatele
   if(max1>max2)
      return max1;
   else
      return max2;
}

int main()
{
   int v[100], n;
   cin>>n;
   for(int i=0; i<n; i++)
      cin>>v[i];
   cout<<maxim(v, 0, n-1);
   return 0;
}
```

Problema 3 (MergeSort)

Se citește de la tastatură un vector cu **n** ($1 \leq n \leq 100$) elemente numere întregi. Să se afișeze vectorul sortat crescător.

MergeSort este o metodă eficientă de sortare a elementelor unui tablou, bazată pe următorul principiu: dacă prima jumătate a unui tablou are elementele sortate, iar a doua jumătate are de asemenea elementele sortate, prin interclasare se obține un tablou sortat.

```
#include <iostream>
using namespace std;

//declaram global vectorii
int v[100], temp[100];

void MergeSort(int v[], int st, int dr)
{
   //problema elementara
   //daca intervalul are un singur element, acesta este deja sortat
   if (st==dr)
      return;

   //se imparte problema in subprobleme
   int mij=(st+dr)/2;
   MergeSort(v, st, mij); //rezolvam prima subproblema
   MergeSort(v, mij+1, dr); //rezolvam a doua subproblema

   //se combina rezultatele
   //aplicam algoritmul de interclasare
   int i=st;
   int j=mij+1;
   int k=st;

   while(i<=mij && j<=dr)
   {
      if(v[i]<=v[j])
         temp[k++] = v[i++];
      else
         temp[k++] = v[j++];
   }

   while(i<=mij)
      temp[k++]=v[i++];

   while(j<=dr)
      temp[k++]=v[j++];

   //se copiaza elementele sortate din temp in vectorul original
   for(i=st; i<=dr; i++)
      v[i]=temp[i];
}
```

```
int main()
{
    int n;
    cin>>n;
    for(int i=0; i<n; i++)
        cin>>v[i];
    MergeSort(v, 0, n-1);
    for(int i=0; i<n; i++)
        cout<<v[i]<<" ";
    return 0;
}
```

Problema 4 (QuickSort)

Se citește de la tastatură un vector cu **n** ($1 \leq n \leq 100$) elemente numere întregi. Să se afișeze vectorul sortat crescător.

Metoda **QuickSort** se bazează pe următorul principiu: se alege un element special al listei, numit **pivot**, apoi se ordonează elementele listei, astfel încât toate elementele din stânga pivotului să fie mai mici sau egale cu acesta, iar toate elementele din dreapta pivotului să fie mai mari sau egale cu acesta. Se continuă recursiv cu secvența din stânga pivotului, respectiv cu cea din dreapta lui.

```
#include <iostream>
using namespace std;
int v[100];

void QuickSort(int v[], int st, int dr)
{
   if(st>=dr)
      return;

   //alegem pivotul
   int mij=(st + dr)/2;
   int aux=v[st];
   v[st]=v[mij];
   v[mij]=aux;

   int i=st , j=dr, d=0;
   while(i<j)
   {
      if(v[i]>v[j])
      {
         aux=v[i];
         v[i]=v[j];
         v[j]=aux;
         d=1-d;
      }
   i+=d;
   j-=1-d;
   }
   QuickSort(v, st, i-1);
   QuickSort(v, i+1, dr);
}

int main()
{
   int n;
   cin>>n;
   for(int i=0; i<n; i++)
      cin>>v[i];
   QuickSort(v, 0, n-1);
   for(int i=0; i<n; i++)
      cout<<v[i]<<" ";
   return 0;
}
```