```
**mergesort**(start, stop)
{
  if start<stop then
  {
    mijloc=(start+stop)/2
    mergesort(start, mijloc)  // sorteaza prima jumatate
    mergesort(mijloc+1, stop)  // sorteaza a doua jumatate
    merge(start, mijloc, stop)  // interclaseaza vectorii sortati
  }
}

merge(start, mijloc, stop)
{
  for i=start to stop
    b[i]=a[i] // copiaza jumatatile sortate intr-un vector auxiliar b

  i=start; j=mijloc+1; k=start;

  // copiaza inapoi in a pe cel mai mic dintre elementele curente in
  // cele 2 jumatati
  while (i<=mijloc) and (j<=stop)
    if b[i]<=b[j] then
      a[k++]=b[i++]
    else
      a[k++]=b[j++]

  // copiaza ce a mai ramas din prima jumatate, daca in ea a mai ramas
  while i<=mijloc
    a[k++]=b[i++]
  // similar pt a doua
  while j<=stop
    a[k++]=b[j++]
}
```

```
cautare_binara(start, stop, n, a)
{
  while start<stop
  {
    mijloc=(start+stop)/2
    if a[mijloc]>n then
      return cautare_binara(start, mijloc, n, a)
    else if a[mijloc]<n then
      return cautare_binara(mijloc, stop, n, a)
    else return true
  }
  return (a[start]==n)
}
```

```
hanoi(n, a, c, b)
{
  if n=1
    move1(a, c)
  else
  {
    hanoi(n-1, a, b, c)
    move1(a, c)
    hanoi(n-1, b, c, a)
  }
}
```

```
factorial(n)
{
  if n<1
    return 1
  else return n*factorial(n-1)
}
```