Exemplu de demonstratie de corectitudine

```
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++]
}
```

Afirmatia 1: algoritmul mergesort este corect (sorteaza vectori intre indicele start si stop) in ipoteza ca procedura merge este corecta.

Se demonstreaza prin **inductie completa** dupa dimensiunea n a vectorului de sortat.

Caz de baza: n=1: In acest caz, in procedura mergesort, nu se intra pe conditia start<stop, iar vectorul este lasat ca atare si algoritmul se termina. Cum era vector de 1 element, era deja sortat, asadar mergesort functioneaza corect pe cazul de baza.

Pas de inductie:

Ipoteza inductiva: mergesort functioneaza corect pe toti vectorii de dimensiune < n.

Trebuie demonstrat ca mergesort functioneaza corect pe vectorii de dimensiune n.

In acest caz, start<stop si se va continua cu impartirea vectorului in 2 si sortarea jumatatilor prin mergesort. Fiecare jumatate va avea dimensiunea<n, asadar mergesort va functiona corect pe ea, conform ipotezei inductive. Cum am presupus ca merge functioneaza corect la randul sau (interclaseaza corect doi vectori sortati pentru a obtine

un vector mare sortat), rezulta ca mergesort va functiona corect si pe vectorul de dimensiune n.

Afirmatia 2: procedura merge este corecta (daca primeste un vector a constand in 2 jumatati sortate intre start si mijloc, respectiv intre mijloc si stop, va produce un vector a cu aceleasi elemente sortate intre start si stop).

Se demonstreaza folosind un **invariant la ciclare**, si anume propozitia: “dupa iteratia i, primele i elemente ale vectorului a sunt sortate”.

Aceasta se demonstreaza prin **inductie matematica** dupa i.

Caz de baza: i=0: primele 0 elemente din a sunt sortate, evident.
Pas de inductie:

Ipoteza inductiva: Dupa iteratia k, primele k elemente din a sunt sortate.

Trebuie demonstrat ca dupa iteratia k+1, primele k+1 elemente din a sunt sortate.

Cum primele k sunt sortate, conform ipotezei inductive, tot ce trebuie sa aratam este ca elementul de pe pozitia k+1 este mai mare sau egal decat toate care il preceda. De aici demonstratia continua simplu, prin deductie directa.

Din combinarea afirmatiilor 1 si 2 rezulta corectitudinea algoritmului mergesort.

Demonstrarea corectitudinii in raport cu tehnica de programare

Divide et impera

- algoritmii divide et impera sunt predispusi la **inductie completa**
- rezolvarea problemei implica rezolvarea unor probleme de dimensiune mai mica (pentru care inductia completa ne garanteaza ca algoritmul este corect)
- in plus, pentru fiecare problema in parte, trebuie sa aratam ca felul in care sunt prelucrate rezultatele problemelor de dimensiune mai mica este corect
- exemplu: mergesort
- observatie: tehnica este aplicabila in general algoritmilor recursivi care se bazeaza pe micsorarea dimensiunii problemei si rezolvarea de probleme din ce in ce mai “simple”

Generate and test

- tot ce trebuie sa faci la generate and test este sa te asiguri ca generezi intr-adevar toate solutiile posibile (nu iti scapa niciuna)
- daca U este spatiul solutiilor partiale, un algoritm corect de tip generate and test va enumera U
- o demonstratie general valida pentru asta este sa **consideram un element arbitrar din U si sa vedem la ce pas este parcurs**
- observatie: daca U este infinita si nu exista solutii, algoritmul nu se va termina (inseamna asta ca nu e corect?)

Exemplu pentru generate and test

Problema: Sa se gaseasca o solutie pentru marea teorema a lui Fermat: $a^n+b^n=c^n$, cu $a,b,c,n \in \mathbf{N}$ si $n>2$.

Alg_Fermat()

1. i=1
2. $a=\log_2(cmmdc(i,2^i))$
3. $b=\log_3(cmmdc(i,3^i))$
4. $c=\log_5(cmmdc(i,5^i))$
5. $n=\log_7(cmmdc(i,7^i))$
6. if $(a,b,c>0)$ and $(a^n+b^n=c^n)$ then return
7. i++
8. goto 2

Spatiul solutiilor partiale este aici **N**x**N**x**N**x**N**. Ceea ce ne trebuie este un generator al tuturor elementelor din acest spatiu. Algoritmul de mai sus parcurge numerele naturale si pentru fiecare identifica puterile la care se gasesc 2, 3, 5 si 7 in descompunerea in factori primi a numarului respectiv. Ideea este ca pentru orice cvartet (a,b,c,n) de numere naturale va exista un numar care sa contina ca factori pe 2^a, 3^b, 5^c si 7^n.

Demonstratia de corectitudine: Alegem arbitrar (a,b,c,n) o solutie din **N**x**N**x**N**x**N** si aratam in ce pas este gasita aceasta solutie. Pasul va fi $i=2^a*3^b*5^c*7^n$.

Exercitiu: insertion-sort

```
insertion-sort(v)
for j=2 to n
   elem<-v[j]
   i<-j-1
   while i>0 and elem<v[i]
      v[i+1]<-v[i]
      i<-i-1
   v[i+1]<-elem
```