n-fakultet uppfyller kraven för rekursion

- Det måste finnas ett basfall av problemet som är enkelt att lösa (dvs utan rekursion):
  Basfallet kan väljas som 1! = 1.
- Det måste finnas ett sätt att förenkla problemet så att det kommer närmare basfallet:
  *Man kan på ett enkelt sätt komma från n! till (n – 1)!, nämligen genom att inse att n! = n * (n - 1)!*

n-fakultet uppfyller kraven för rekursion

- Det måste finnas ett basfall av problemet som är enkelt att lösa (dvs utan rekursion):
  Basfallet kan väljas som 1! = 1.
- Det måste finnas ett sätt att förenkla problemet så att det kommer närmare basfallet:
  *Man kan på ett enkelt sätt komma från n! till (n – 1)!, nämligen genom att inse att n! = n * (n - 1)!*

```
faq(0,Y):- Y is 1.
faq(X,Y):- T is X-1,faq(T,P),Y is X*P.
```

Metod för att summera talen i en vektor

- Rekursiv metod:

```
public int sum(int[] a, int startPos)
      {  if (startPos >= a.length)
            return 0;
      else
            return a[startPos] + sum(a, startPos + 1);

}
```

Metod för att summera talen i en vektor

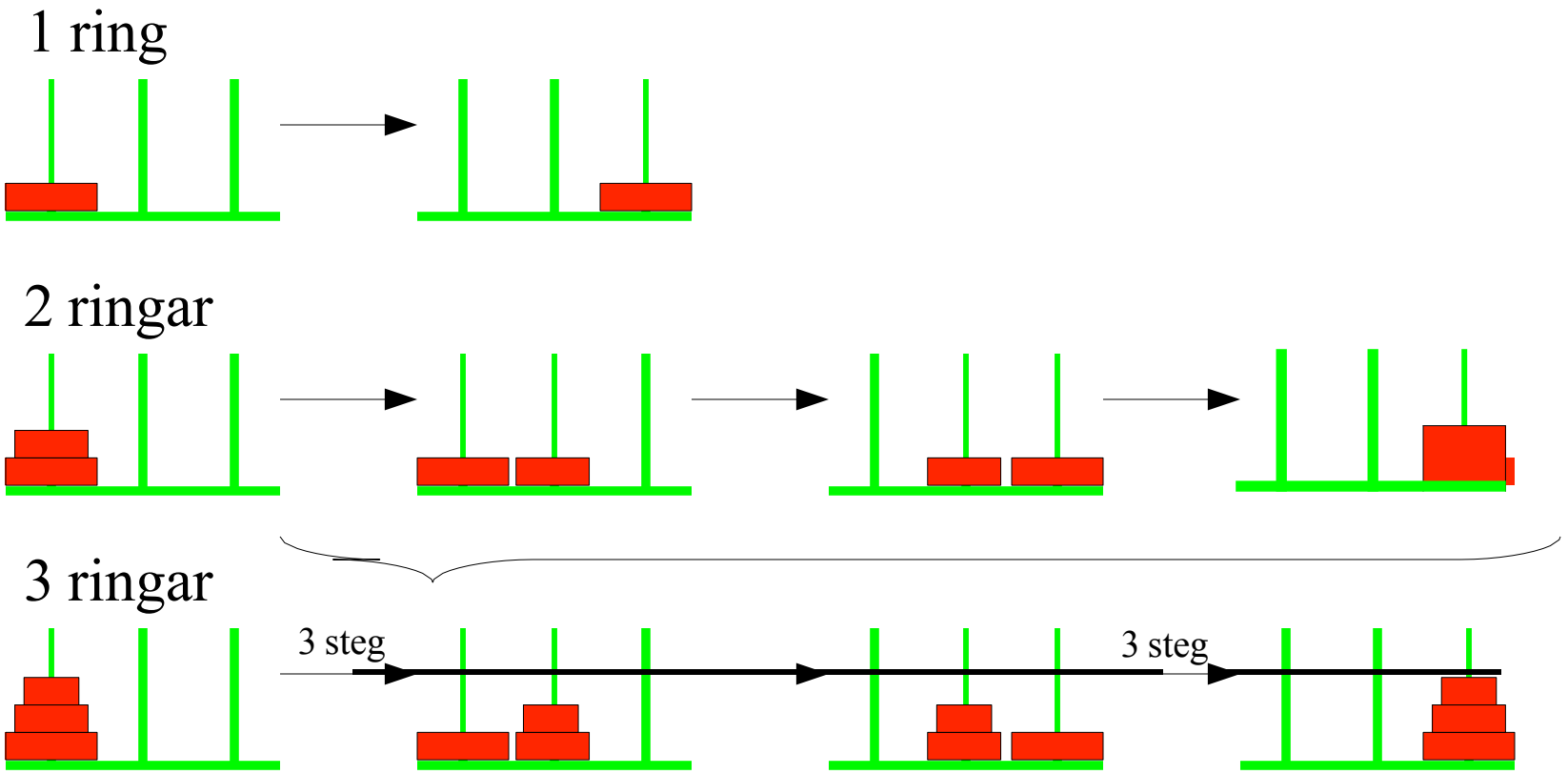
- Rekursiv metod:

```
public int sum(int[] a, int
      startPos) {  if (startPos >=
      a.length)
            return 0;
      else
            return a[startPos] + sum(a, startPos
}           + 1);
```

Prolog:

```
sum([],Y):- Y is 0.
sum([X|T],Y):- sum(T,P),Y is X+P.
```

Tornen i Hanoi



Tornen i Hanoi uppfyller kraven för rekursion

- Problemet ska bestå av en eller flera delproblem som liknar problemet självt:
  Problemet att flytta en hög med ringar innehåller delproblemet att flytta en hög med färre antal ringar.
- Det måste finnas ett basfall av problemet som är enkelt att lösa (dvs utan rekursion):
  Basfallet utgörs av problemet med 1 ring. Då är det självklart vad man ska göra: Flytta ringen till höger pinne.
- Det måste finnas ett sätt att förenkla problemet så att det kommer närmare basfallet:
  För att flytta n ringar från pinne A till B, gör så här:
  - *flytta n-1 ringar från pinne A till annanPinne(A,B)*
  - *flytta ring n till pinne B*
  - *flytta n-1 ringar från annanPinne(A,B) till B*

Tornen i Hanoi uppfyller kraven för rekursion

- move(1,X,Y,_) :- write('Move top disk from '), write(X),
  write(' to '), write(Y), nl.

- move(N,X,Y,Z) :- N>1, M is N-1,
  move(M,X,Z,Y),
  move(1,X,Y,_),
  move(M,Z,Y,X).

Tornen i Hanoi uppfyller kraven för rekursion

```
towersOfHanoi(N,A,B,C,A4,B4,C4) :-  move(N,A,B,C,A4,B4,C4),!.

move(1,[H|T],B,C,A1,B1,C1) :- A1 = T,
                              B1 = [H|B],
                              C1 = C.

 move(N,A,B,C,A4,B4,C4) :- N>1, M is N-1,
                           move(M,A,C,B,A1,C1,B1),
                           move(1,A1,B1,C1,A2,B2,C2),
                           move(M,C2,B2,A2,C4,B4,A4).
```

Rekursiva sorteringsmetoder

- **Quicksort.** Denna sorteringsalgoritm är själv-upprepande och har följande steg:
  1) Välj ett element, kallat pivotelementet, ur listan.
  2) Ordna om listan så att alla element som är mindre än pivotelementet kommer före detta, och så att alla element som är större än pivotelementet kommer efter detta. Pivotelementet har nu fått sin rätta plats.
  3) Sorter rekursivt de två dellistorna, dvs med just denna algoritm.

	5	13	2	8	11	3	17	9	1
1)	5	13	2	8	11	3	17	9	1
2)	2	3	1	5	13	8	11	17	9
3)	1	2	3	5	8	9	11	13	17

Lista att sortera.

Första elementet väljs som pivotelement.

Pivotelementet placeras på rätt plats.

Quicksort.

```
algorithm quicksort(A, lo, hi) is
        if lo < hi then p := partition(A, lo, hi)
        quicksort(A, lo, p - 1 )
        quicksort(A, p + 1, hi)

algorithm partition(A, lo, hi) is
        pivot := A[hi] i := lo - 1
        for j := lo to hi - 1 do
        if A[j] < pivot then
        i := i + 1
        swap A[i] with A[j]
        swap A[i + 1] with A[hi]
return i + 1
```

Quicksort.

```
quicksort([X|Xs],Ys) :-
        partition(Xs,X,Left,Right),
        quicksort(Left,Ls),
        quicksort(Right,Rs),
        append(Ls,[X|Rs],Ys).

quicksort([],[]).

partition([X|Xs],Y,[X|Ls],Rs) :-
                X <= Y, partition(Xs,Y,Ls,Rs).
partition([X|Xs],Y,Ls,[X|Rs]) :-
                X > Y, partition(Xs,Y,Ls,Rs).
partition([],Y,[],[]).

append([],Ys,Ys).
append([X|Xs],Ys,[X|Zs]) :- append(Xs,Ys,Zs).
```