Kilde: http://www.docjava.dk/grundlaeggende\_programmering/rekursion/rekursion.htm

|  |  |
| --- | --- |
| **Euclid** | Euclid (300 fvt.) er ophavsmand til en meget kraftfuld algoritme til at finde den største fælles divisor (det største tal der går op i dem begge). Metoden er rekursiv og givet ved: |
|  | |  | | --- | | gcd( m, n ) = gcd( n, m modulus n ), hvis n ikke går op i m  gcd( m, n ) = n, ellers | |
|  | hvor man antager: m >= n > 0. Man navngiver traditionelt funktionen, der beregner største fælles divisor: **gcd** (**g**reatest **c**ommon **d**ivider). |
|  | Basis er, når n går op i m; hvor resultatet blot er n. Rekursions-skridtet er et enkelt kald videre. Man bemærker at der om **m modulus n** gælder: **n > m modulus n > 0**. Derfor passer antagelsen om m >= n > 0 også i kaldet videre. |
|  | Den rekursive metode bliver: |
| Source 6:*Rekursiv: Største fælles divisor* | |  | | --- | | static int gcd( int m, int n ) {  // PRE: m >= n > 0  if ( m%n == 0 )  return n;  else  return gcd( n, m%n );  } | |
|  | Lad os igen tegne et diagram der viser de rekursive kalds sammenhæng. Vi vil finde den største fælles divisor for 21 og 12: |
| Figur 3: *Forløbet af de rekursive kald* | |  | | --- | | http://www.docjava.dk/grundlaeggende_programmering/rekursion/rekursion_gcd.gif | |
|  | Diagrammet ligner i sin struktur det vi tegnede for fakultet. Det skyldes at de begge arbejder med ét rekursivt kald videre, mens f.eks. fibonacci bruger to kald og dermed får en anderledes struktur. |
|  | Den iterative løsning bliver: |
| Source 7:*Iterativ: Største fælles divisor* | |  | | --- | | static int gcd( int m, int n ) {  // PRE: m >= n > 0  int tmp;  while ( m%n > 0 ) {  tmp = n;  n = m%n;  m = tmp;  }  return n;  } | |
|  | Den rekursive er en anelse enklere, men forskellen er igen ubetydelig. |
|  |  |

Vigtigt: En rekursiv algoritme kan altid omskrives til en iterativ algoritme, men det omvendte gælder IKKE. En iterativ algoritme kan ikke altid omskrives til en rekursiv algoritme