Implementasi Fungsi Rekursif Dalam Algoritma dan Perbandingannya dengan Fungsi Iteratif

Stephen Herlambang - 13507040
Jurusan Teknik Informatika ITB, Bandung 40116, email: norbert@students.itb.ac.id

Abstract *– Makalah ini membahas implementasi fungsi rekursif dalam algoritma dan perbandingannya dengan fungsi iteratif. Fungsi rekursif merupakan fungsi yang melakukan perulangan dengan mengacu pada dirinya sendiri, sedangkan fungsi iteratif merupakan fungsi yang melakukan perulangan terutama dengan menggunakan loop. Fungsi rekursif maupun fungsi iteratif dapat diimplementasikan dalam berbagai algoritma dengan kelebihan dan kekurangannya masing-masing.*

Kata Kunci*: fungsi, rekursi, rekursif, iterasi, iteratif, algoritma.*

1. PENDAHULUAN

Rekursif adalah salah satu metode dalam dunia matematika dimana definisi sebuah fungsi mengandung fungsi itu sendiri. Dalam dunia pemrograman, rekursi diimplementasikan dalam sebuah fungsi yang memanggil dirinya sendiri.
Sebenarnya, rekursi merupakan salah satu cara berpikir / pola pikir untuk menyelesaikan masalah. Hal ini memang merupakan ide / tujuan utama dalam sains komputer. Menyelesaikan masalah menggunakan rekursi berarti solusi bergantung pada solusi masalah yang sama dalam bentuk yang lebih tidak kompleks.
Iterasi merupakan suatu cara untuk menyelesaikan masalah dengan melakukan perulangan di dalam fungsi itu sendiri. Iterasi mendeskripsikan gaya pemrograman yang digunakan pada bahasa pemrograman *imperative*.
Kebanyakan bahasa pemrograman mendukung konsep rekursi ini dengan memperbolehkan suatu fungsi memanggil dirinya sendiri dan juga mendukung konsep iteratif. Meskipun begitu, ada pula bahasa pemrograman yang tidak bisa melakukan salah satu dar kedua jenis perulangan tersebut.

2. REKURSI

Rekursi adalah cara untuk menetapkan proses dengan dirinya sendiri
Sebagai contoh, berikut ini adalah definisi rekursif dari seorang leluhur seseorang:
Orang tua seseorang adalah leluhur orang tersebut. (Basis kasus)
Orang tua dari leluhur seseorang adalah juga leluhur orang tersebut. (Langkah rekursif)
Contoh cara yang lebih mudah untuk mengerti proses rekursi adalah sebagai berikut:
1. Apakah kita telah selesai? Jika iya, kembalikan hasilnya. Tanpa terminasi seperti ini, maka proses rekursi akan berlangsung terus-menerus.
2. Jika tidak, simpilifikasikan masalah tersebut, dan kumpulkan hasilnya pada solusi untuk masalah awal. Kemudian kembalikan hasil tersebut. [7]

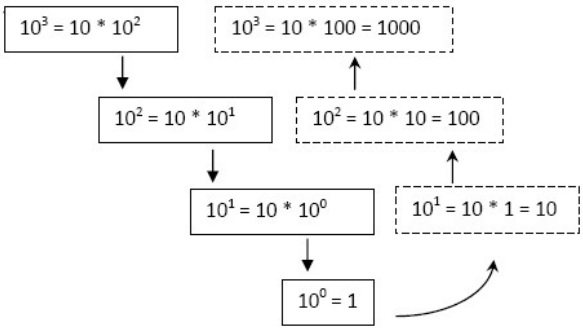
2.1. Fungsi Rekursif

Fungsi merupakan salah satu jenis relasi. Misalkan terdapat himpunan A sebagai *domain* dan himpunan B sebgai *codomain*, maka suatu relasi disebut fungsi jika relasi tersebut menghubungkan setiap elemen di dalam A ke tepat satu elemen di dalam B. [1]
Fungsi rekursif adalah fungsi yang mengacu pada dirinya sendiri. Fungsi rekursif disusun atas dua bagian, yaitu basis dan rekurens.
Basis berisi nilai awal yang tidak mengacu pada dirinya sendiri. Bagian ini berfungsi untuk memberikan nilai yang terdefinisi pada fungsi rekursif dan sekaligus menghentikan proses rekursi.
Rekurens merupakan bagian fungsi yang mendefinisikan argumen fungsi dalam terminologi dirinya sendiri. Setiap kali fungsi tersebut melakukan bagian rekurens maka argumen dari fungsi tersebut harus lebih dekat pada basisnya.
Contoh penggunaan fungsi rekursif adalah sebagai berikut:
Dalam fungsi x^y, untuk semua bilangan selain 0, jika dipangkatkan dengan 0 nilainya sama dengan 1. Jika x dipangkatkan dengan y, dengan y lebih dari 0, maka hasilnya sama dengan x dikalikan dengan x dipangkatkan y – 1. Jika dituliskan dalam notasi matematika definisinya adalah sebagai berikut:

$$f(x,y) = \begin{cases} 1 & , y = 0 \\ x.f(x,y-1) & , y > 0 \end{cases}$$

Dari definisi di atas dapat dilihat bahwa pada definisi y > 0, bentuk pemangkatan muncul kembali di sisi kanan. Itulah yang disebut bagian rekurens. Definisi rekursif selalu dimulai dengan kasus penyetop, penghenti, atau kasus dasar dari suatu permasalahan, dalam hal ini terjadi ketika nilai y = 0, bagian inilah yang disebut basis.
Definisi rekursif yang lebih kompleks mengandung inti dari permasalahan yang akan dipecahkan, namun lebih sederhana. Dalam hal ini yang tadinya x dipangkatkan dengan y, kini bentuk pemangkatan menjadi lebih sederhana, yaitu y-1. Hal ini dimaksudkan untuk “menggiring” masalah kompleks

ke kasus dasar atau penyetop rekursinya (basis). [3]
Untuk x = 10 dan y = 0, hasil dari x^y adalah 1. Untuk x = 10 dan y = 3 hasilnya dapat digambarkan sebagai berikut:



Gambar 1: Proses pada fungsi x^y dimana x=10 dan y=3

Contoh lain fungsi yang dapat dibuat dengan menggunakan konsep rekursif adalah fungsi faktorial:

$$f(n) = n!$$

Definisi rekursifnya adalah:

$$f(n) = \begin{cases} 1 & ,n = 0 \\ n \,.\, f(n-1), & n > 0 \end{cases}$$

Basis dari fungsi ini adalah :

$$f(n) = 1, n = 0$$

Rekurens dari dari fungsi ini adalah:

$$f(n) = n \,.\, f(n-1), n > 0$$

2.2. Fungsi Rekursif dalam Algoritma

Dalam dunia pemrograman, rekursi dapat diimplementasikan dalam fungsi yang memanggil dirinya sendiri.
Konsep pemrograman fungsi rekursif sebenarnya mirip dengan konsep fungsi rekursif dalam matematika yaitu sebagai berikut:
1. Menentukan kasus penyetop atau kasus dasar dimana pemanggilan rekursif tidak lagi diperlukan karena solusinya sudah diperoleh) -> basis
2. Menerapkan suatu langkah untuk menggiring kasus kompleks ke kasus penyetopnya dengan metode yang mencerminkan fungsinya -> rekurens [3]

Contoh implementasi funsi x^y pada algoritma dengan *pseudo-code* dan menggunakan konsep rekursif adalah sebagai berikut:

```
function Pangkat (x : integer, y : integer) -> integer
DEKLARASI
ALGORITMA
    if (y = 0)
        -> 1
    else
        -> x * Pangkat(x,y-1)
```

Gambar 2: Algoritma rekursif untuk fungsi x^y

Contoh implementasi fungsi faktorial pada algoritma dengan *pseudo-code* adalah sebagai berikut:

```
function Factorial (n : integer) -> integer
DEKLARASI
ALGORITMA
    if (n = 0) then (Basis)
        -> 1
    else (Rekurens)
        -> n * (Factorial (n-1))
```

Gambar 3: Algoritma rekursif untuk fungsi Factorial

3. ITERASI

Iterasi berarti suatu aksi yang berulang/ perulangan.
Perulangan yang dilakukan pada iterasi berbeda dengan perulangan pada rekursi. Perulangan pada iterasi dilakukan di dalam fungsi itu sendiri.

3.1. Fungsi Iteratif dalam Algoritma

Iterasi dalam algoritma adalah proses perulangan dalam suatu prosedur atau fungsi, terutama dalam bentuk *loop*.
Fungsi ini memiliki ciri dimana adanya nilai variabel yang terus berubah selama terjadi perulangan. [5]
Fungsi iteratif berarti fungsi yang menggunakan konsep iterasi dalam prosesnya.
Dalam makalah ini tidak dibahas prosedur melainkan hanya dibahas mengenai fungsi iterasi. Hal ini dikarenakan untuk perbandingan dengan fungsi rekursif.
Contoh fungsi x^y yang diimplementasikan dengan menggunakan konsep iteratif adalah sebagai berikut:

```
function Pangkat (x : integer, y : integer) -> integer
DEKLARASI
    i : integer
    sum : integer
ALGORITMA
    i <- 1
    sum <- 1
    while (i <= y) do
        sum <- sum*x
        i <- i+1
    -> sum
```

Gambar 4: Algoritma iteratif untuk fungsi x^y

Contoh fungsi iteratif dalam algoritma *pseudo*-code untuk fungsi faktorial adalah sebagai berikut:

```
function Factorial (n : integer) -> integer
DEKLARASI
    i, sum : integer
ALGORITMA
    i <- n
    sum <- 1
    while (i > 1) do
        sum <- sum*i
        i <- i-1
    -> sum
```

Gambar 5: Algoritma iteratif untuk fungsi Factorial

4. IMPLEMENTASI FUNGSI REKURSIF DAN ITERATIF SERTA PERBANDINGANNYA

4.1. Implementasi untuk Fibonacci

Deret Fibonacci merupakan deretan angka dengan 20 angka pertama sebagai berikut:

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

Gambar 6: Deret Fibonacci untuk 20 angka pertama

Fungsi fibonacci secara rekursif dapat didefinisikan sebagai berikut: [4]

$$f(n) = \begin{cases} 0 & ,n = 0 \quad (basis) \\ 1 & ,n = 1 \quad (basis) \\ f(n-1) + f(n-2), n \geq 2 & (rekurens) \end{cases}$$

Konsep rekursif dari definisi fungsi tersebut dapat diimplementasikan dalam *pseudo-code* dengan algoritma sebagai berikut:

```
function Fibonacci (n : integer) -> integer
DEKLARASI
ALGORITMA
    if (n = 0) then
        -> 0
    else
        if (n = 1) then
            -> 1
        else
            -> (Fibonacci(n-1) + Fibonacci(n-2))
```

Gambar 7: Algoritma rekursif untuk fungsi Fibonacci

Atau dapat juga disederhanakan sebagai berikut:

```
function Fibonacci (n : integer) -> integer
DEKLARASI
ALGORITMA
    if (n <= 2) then
        -> 1
    else
        -> (Fibonacci(n-1) + Fibonacci(n-2))
```

Gambar 8: Algoritma rekursif fungsi Fibonacci versi 2

Sedangkan implementasi dengan menggunakan konsep iteratif adalah sebagai berikut:

```
function Fibonacci (n : integer) -> integer
DEKLARASI
    fib : array[1..(n+1)] of integer
    i : integer
ALGORITMA
    fib1 <- 1
    fib2 <- 1
    i <- 3
    while (i <= n) do
        fi <- fi-1 + fi-2
        i <- i+1
    -> fn
```

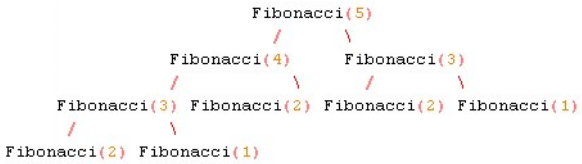
Gambar 9: Algoritma iteratif untuk fungsi Fibonacci

Algoritma ini menggunakan *array* sebagai penyimpan nilai (akumulator). Dengan begitu, algoritma ini banyak menghabiskan tempat di memori.
Versi lain dari implementasi menggunakan konsep iteratif adalah sebagai berikut:

```
function Fibonacci (n : integer) -> integer
DEKLARASI
    a, b, c : integer
ALGORITMA
    a <- 1
    b <- 1
    i <- 3
    while (i <= n) do
        c <- a + b
        a <- b
        b <- c
    -> b
```

Gambar 10: Algoritma iteratif fungsi Fibonacci versi 2

Dengan melihat implementasi algoritma antara menggunakan konsep rekursif dan konsep iteratif, dapat dilihat implementasi dengan menggunakan konsep rekursif jauh lebih mudah dipahami dan dibuat. Hal ini dikarenakan implementasi fungsi yang dibuat persis mengikuti definisi dari fungsi rekursif tersebut.
Selain itu, pada versi yang menggunakan konsep iteratif, digunakan variabel tambahan untuk penyimpanan nilai. Variabel ini juga harus diinisialisikan dahulu. Hal ini berarti tambahan penggunaan memori serta proses pada komputer.
Tetapi, proses secara rekursif juga bukannya tidak membutuhkan banyak penggunaan memori. Hal ini disebabkan adanya proses berulang yang seakan-akan terus-menerus menyimpan nilai pada stack yang tidak ada.
Hal ini dapat digambarkan dengan misalnya proses penggunaan fungsi Fibonacci ini untuk integer n = 5:



Gambar 11: Algoritma rekursif untuk fungsi Fibonacci

Untuk n = 5 saja, dilakukan 9 kali pemanggilan terhadap fungsi rekursif Fibonacci ini. Masing-masing pemanggilan melakukan proses dan juga mengembalikan nilai untuk kemudian diproses lagi pada pemanggilan fungsi di bawahnya sampai kondisi berhenti. Hal ini berarti menambah penggunaan memori pada komputer.
Sedangkan untuk versi iteratif tentu saja hanya dilakukan 1 kali pemanggilan terhadap fungsi tersebut sebab pengulangan dilakukan di dalam fungsi, hal ini mempersedikit penggunaan memori.

4.2. Implementasi untuk fungsi McCarthy 91

Fungsi McCarthy merupakan fungsi yang menghasilkan nilai 91 untuk setiap n < 101 dan n-10 untuk setiap n > 100. [6]
Fungsi tersebut dapat didefinisikan secara rekursif sebagai berikut:

$$M(n) = \begin{cases} n - 10 & ,n > 100 \quad (basis) \\ M(M(n+11)), n \leq 100 & (rekurens) \end{cases}$$

Implementasi dalam pseudocode adalah algoritma

sebagai berikut:

```
function McCarthy (n : integer) -> integer
DEKLARASI
ALGORITMA
    if (n > 100) then (basis)
        -> n-10
    else (rekurens)
        -> McCarthy(McCarthy(n+11))
```

Gambar 12: Algoritma rekursif untuk fungsi McCarthy 91

Contoh penggunaan:
McCarthy(99)= McCarthy (McCarthy (110))
= McCarthy (100) = McCarthy (McCarthy (111))
= McCarthy (101) = 91
McCarthy(88) = McCarthy(McCarthy(99))
= McCarthy(91) = McCarthy(McCarthy(102))
= McCarthy(92) = McCarthy(McCarthy(103))
dst.
= McCarthy(99) = 91

Fungsi McCarthy 91 ini juga dapat diimplementasikan dalam algoritma dengan menggunakan konsep iteratif. Contoh implementasinya adalah sebagai berikut:

```
function McCarthy (n : integer) -> integer
DEKLARASI
    i, a : integer
ALGORITMA
    i <- 1
    a <- n
    while (i ≠ 0)
        if (a > 100) then
            a <- a-10
            i <- i-1
        else
            a <- a+11
            i <- i+1
    -> n
```

Gambar 13: Algoritma iteratif untuk fungsi McCarthy 91

Pada perbandingan antara fungsi McCarthy 91 menggunakan konsep rekursif dan konsep iteratif ini juga dapat dilihat bahwa fungsi dengan konsep rekursif jauh lebih mudah dipahami dan diterapkan.
Tetapi, penggunaan memori pada versi rekursif jauh lebih banyak, karena perlu tambahan untuk semacam stack untuk menyimpan hasil setiap pemanggilan fungsi. Selain itu, waktu yang diperlukan untuk melakukan proses juga lebih lama karena proses yang dilakukan juga lebih banyak, yaitu perlu menjejaki setiap pemanggilan rekursif melalui sampai basis, baru kemudian diproses kembali untuk operasi sebelumnya yang tertunda.

4.3. Implementasi untuk Type Bentukan List

Misalkan terdapat tipe bentukan List dengan 2 **isi** yaitu Info dan Next. Info merupakan isi dari address List tersebut yang dapat diakses dengan List.Info dan Next merupakan *pointer* yang mengakses address selanjutnya yang dapat diakses dengan List.Next. List tersebut mengacu pada address di elemen First. List kosong bernilai Nil dan elemen terakhir List memiliki Next = Nil. [2]
Misalkan diperlukan fungsi untuk mengakses Info dari elemen List ke n.
Fungsi tersebut dapat didefinisikan sebagai berikut:

$$f(L,n) = \begin{cases} L.info & ,n = 1 \quad (basis) \\ f(L.next, n-1) & ,n > 1 \ (rekurens) \end{cases}$$

Implementasi fungsi tersebut dengan konsep rekursif adalah seperti berikut:

```
function InfoKeN (L : List, n : integer) -> infotype
DEKLARASI
ALGORITMA
    if (n = 1) then (basis)
        -> L.Info
    else (rekurens)
        -> InfoKeN(L.Next,n-1)
```

Gambar 14: Algoritma rekursif untuk fungsi InfoKeN

Sedangkan salah satu cara implementasi secara iteratif dari fungsi tersebut adalah sebagai berikut:

```
function InfoKeN (L : List, n : integer) -> infotype
DEKLARASI
    i : integer
    CL : address
ALGORITMA
    CL <- L
    i <- n
    while (i > 1) do
        CL <- CL.Next
        i <- i-1
    -> CL.Info
```

Gambar 15: Algoritma iteratif untuk fungsi InfoKeN

Selain itu, misalnya juga diperlukan fungsi untuk menghitung jumlah elemen yang memiliki Info = X pada List tersebut, maka fungsi tersebut dapat diimplementasikan sebagai berikut:

```
function JumlahX (L : List, X : infotype) -> integer
DEKLARASI
ALGORITMA
    if (L = Nil) then
        -> 0
    else
        if (L.Info = X) then
            -> 1 + JumlahX(L.Next,X)
        else
            -> JumlahX(L.Next,X)
```

Gambar 16: Algoritma rekursif untuk fungsi JumlahX

Sedangkan implementasi versi iteratifnya adalah sebagai berikut:

```
function JumlahX (L : List, X : infotype) -> integer
DEKLARASI
    n : integer
    CL : address
ALGORITMA
    n <- 0
    CL <- L
    while (CL ≠ Nil) do
        if (L.Info = X) then
            n <- n+1
        else (do nothing)
        CL <- CL.Next
    -> n
```

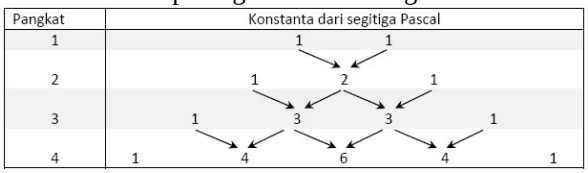
Gambar 17: Algoritma iteratif untuk fungsi JumlahX

Dari kedua contoh implementasi fungsi untuk type bentuan List di atas, dapat dilihat bahwa implementasi dengan konsep rekursif lebih ringkas serta lebih mudah dipahami.
Perbedaan terlihat pada implementasi rekursif

InfoKeN dan JumlahX. Pada InfoKeN, tidak terdapat penumpukan hasil yang berarti tidak diperlukan tambahan memori. Hal ini berarti proses yang dilakukan mirip dengan proses iteratif. Proses rekursif seperti ini disebut juga fungsi *"Tail Recursive"*.
Sedangkan pada JumlahX, proses penumpukan terjadi pada bagian rekurens bila L.Info = X. Pada saat itu, diperlukan penjejakan pemanggilan fungsi sampaid engan basisnya, baru kemudian dikembalikan lagi nilainya untuk diproses pada fungsi yang tertunada sebelumnya. Hal ini berarti tambahan penggunaan memori, dan juga untuk alasan yang sama maka waktu pemrosesan yang diperlukan lebih lama.

4.4. Implementasi dalam $(a+b)^n$

Pola konstanta dari $(a+b)^n$ memiliki pola yang sama dengan pola dari segitiga Pascal.
Pola tersebut dapat digambarkan sebagai berikut:



Gambar 18: Pola konstanta dari $(a+b)^n$

Dengan memperhatikan pola di atas, dapat dilihat bahwa konstanta paling kiri dan paling kanan adalah 1. Sisanya merupakan penjumlahan dari konstanta di atasnya. Angka 2 pada pangkat 2 didapatkan dari 1 + 1, angka 3 pada pangkat 3 diperoleh dari 1 + 2 dan seterusnya.
Pola tersebut dapat diilustrasikan seperti berikut ini:
Pada pangkat ke-1, angka ke-1 = 1, angka ke-2 = 1
Pada pangkat ke-2, angka ke-1 = 1, angka ke-2 = 2, angka ke-3 = 1, dan seterusnya,
Dengan begitu, dapat dibentuk suatu pola rekursif dengan menggunakan fungsi f(x,y) dimana f(x,y) merupakan nilai dari angka ke-y pada pangkat ke-x.

$$f(x,1) = 1$$
$$f(x,x+1) = 1$$
$$f(x,y) = f(x-1,y-1) + f(x-1,y)$$

Pola tersebut dapat didefinisikan dalam suatu fungsi rekursif yaitu sebagai berikut:

$$f(x,y) = \begin{cases} 1 & ,y = 1 \quad (basis) \\ 1 & ,y = x+1 \quad (basis) \\ f(x-1,y-1) + f(x-1,y), 1 < y < x+1 & (rekurens) \end{cases}$$

Implementasi dari fungsi rekursif tersebut dalam algoritma dengan *pseudo-code* adalah sebagai berikut:

```
function Konstanta(x:integer, y:integer) -> integer
DEKLARASI
ALGORITMA
    if (y = 1) or (y = (x+1)) then {basis}
        -> 1
    else {rekurens}
        -> Konstanta(x-1,y-1)+Konstanta(x-1,y)
```

Gambar 19: Algoritma rekursif untuk fungsi Konstanta

Sedangkan salah satu contoh implementasi dengan menggunakan konsep iteratif adalah sebagai berikut:

```
function Konstanta(x:integer, y:integer) -> integer
DEKLARASI
    i, j : integer
    hasil : integer
ALGORITMA
    hasil <- 1
    i <- 1
    j <- 0
    while ((j <= x) and (i < y)) do
        hasil = hasil*(x-j)/(j+1)
        i <- i+1
        j <- j+1
    -> hasil
```

Gambar 20: Algoritma iteratif untuk fungsi Konstanta

Dari kedua versi di atas, dapat dilihat bahwa pada versi rekursif, penurunan konsep dapat dengan lebih mudah dilakukan dan dipahami dibandingkan dengan versi iteratif.
Tetapi kelemahan yang ada pada versi rekursif ini tetap sama, yaitu banyaknya memori yang diperlukan. Hal ini disebabkan alasan yang sama dengan proses pemanggilan pada versi rekursif Fibonacci, yaitu pada setiap pemanggilan fungsi, fungsi tersebut akan terus menerus memanggil fungsi tersebut lagi bila belum mencapai basis yang nilainya harus disimpan untuk kemudian digunakan pada proses yang tertunda sebelumnya, dan hal ini menambah penggunaan memori.

4.5. Implementasi dalam GCD (Greatest Common Divisor)

Fungsi gcd merupakan fungsi untuk mendapatkan niali faktor persekutuan terbesar antara dua buah *integer*. [8]
Fungsi ini memanfaatkan algoritma Euclidean dimana fungsi tersebut dapat didefinisikan sebagai berikut:

$$gcd(x,y) = \begin{cases} x & ,y = 0 \quad (basis) \\ f(y, x \bmod y) & ,x \geq y, y > 0 \quad (rekurens) \end{cases}$$

Implementasi fungsi rekursif tersebut dalam *pseudo-code* adalah sebagai berikut:

```
function gcd (x:integer, y:integer) -> integer
DEKLARASI
ALGORITMA
    if (y = 0) then {basis}
        -> x
    else {rekurens}
        -> gcd(y, x mod y)
```

Gambar 21: Algoritma rekursif untuk fungsi gcd

Sedangkan implementasi untuk versi iteratif adalah sebagai berikut:

```
function gcd (x:integer, y:integer) -> integer
DEKLARASI
    remainder : integer
    i, j : integer
ALGORITMA
    i <- x
    j <- y
    while (j !=0)
        remainder = i mod j
        i = j
        j = remainder
    -> i
```

Gambar 22: Algoritma iteratif untuk fungsi gcd

Setelah melihat versi rekursif dan versi iteratif dari fungsi gcd, terlihat bahwa versi iteratif lebih sukar untuk dipahami. Padahal langkah yang digunakan hampir sama. Selain itu, juga diperlukan variabel tambahan pada versi iteratif. Hal ini berarti menambah penggunaan memori.
Setelah diperhatikan , versi rekursif dari fungsi gcd ini juga merupakan fungsi *"Tail-Recursive"*. Dengan begitu, tidak terjadi penumpukan pemanggilan fungsi ni sampai basis. Pemanggilan fungsi ini hanya berulang terus sampai basis dan kemudian langsung mengembalikan nilainya. Hal ini berarti penggunaan memori yang diperlukan serta waktu yang diperlukan untuk proses mirip dengan proses iteratif.

4.6. Perbandingan Secara Umum Antara Fungsi Rekursif dan Iteratif

Berikut adalah perbandingan secara umum algoritma yang dibuat dengan menggunakan konsep rekursif dan iteratif:

Tabel 1: Perbandingan umum fungsi rekursif dan iteratif

Fungsi rekursif	Fungsi iteratif
Algoritma biasanya lebih ringkas dan mudah dipahami	Kode program lebih panjang, untuk beberapa kasus solusi iteratif lebih sulit diterapkan
Membutuhkan alokasi memori yang besar (kecuali pada fungsi *"Tail-Recursive"*)	Relatif lebih kecil alokasi memorinya
Tidak cocok ketika kinerja tinggi diperlukan, karena terjadi *overhead* pemanggilan fungsi dalam jumlah yang relatif besar	Cocok diterapkan ketika kinerja aplikasi harus diterapkan (hanya ada satu kali pemanggilan fungsi)

5. KESIMPULAN

Fungsi rekursif merupakan salah satu konsep yang penting untuk dikuasai. Hal ini dikarenakan fungsi ini memiliki sangat banyak implementasi, yang dalam hal ini adalah implementasi dalam algoritma.
Fungsi rekursif dan fungsi iteratif memiliki kelebihan dan kelemahan masing-masing. Secara umum fungsi rekursif memiliki algoritma yang lebih mudah dipahami dan dibuat tetapi membutuhkan penggunaan memori yang besar.

DAFTAR REFERENSI

[1] R. Munir, *Struktur Diskrit*, ITB, 2008
[2] I. Liem, *Diktat Struktur Data*, ITB, 2008
[3] http://images.wishknew.multiply.com/attachment/0/RnVOEAoKCrgAADq0T381/Rekursif.pdf?nmid=46507331
Waktu akses: 2 Januari 2009, 02.58 PM
[4] http://en.wikipedia.org/wiki/Fibonacci_number
Waktu akses: 1 Januari 2009, 06.25 PM
[5] http://en.wikipedia.org/wiki/Iteration
Waktu akses: 2 Januari 2009, 03.26 PM
[6] http://en.wikipedia.org/wiki/McCarthy_91_function
Waktu akses: 1 Januari 2009, 06.21 PM
[7] http://en.wikipedia.org/wiki/Recursion
Waktu akses: 1 Januari 2009, 06.57 PM
[8] http://en.wikipedia.org/wiki/Recursion_%28computer_science%29
Waktu akses: 1 Januari 2009, 06.21 PM
[9]http://www.daniweb.com/forums/thread68837.html
Waktu akses : 2 Januari 2009, 10.25 PM
[10]http://www.ics.uci.edu/~eppstein/161/960109.htm
Waktu akses: 2 Januari 2009, 04.25 PM