Optimasi Perhitungan Bilangan Fibonacci Menggunakan Program Dinamis

Yudi Retanto 13508085
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
yudiretanto@gmail.com

ABSTRAK

Bilangan Fibonacci adalah angka-angka yang berasal dari deret Bilangan Fibonacci. Bilangan ini sering digunakan untuk menyelesaikan beberapa permasalahan matematika. Secara nyata, bilangan Fibonacci biasa digunakan untuk analisis dalam *financial markets*, dalam strategi seperti *Fibonacci retracement*, dan juga digunakan dalam algoritma pemrograman speri *Fibonacci search technique* dan *Fibonacci heap data structure*. Namun bilangan Fibonacci seringkali dikaitkan dengan hal-hal yang ada di dunia ini seperti jumlah perkembang-biakan kelinci mengikuti deret bilangan Fibonacci, ataupun jumlah daun pada bunga (petals). Pada makalah ini tidak akan membahas mengenai penggunaan bilangan Fibonacci pada maslaah-masalah tersebut, melainkan bagaimana mendapatkan bilangan Fibonacci dengan algoritma pemrograman. Salah satu algortima yang digunakan adalah menggunakan metode rekursif untuk mendapatkan bilangan Fibonacci. Pada kuliah Strategi Algoritma, terdapat beberapa algoritma yang dapat digunakan untuk meningkatkan performa algoritma suatu permasalahan. Penulis mencoba untuk menggunakan algoritma pemrograman dinamis untuk mencoba meningkatkan performa algoritma bilangan Fibonacci. Dengan membandingkan waktu yang digunakan untuk menjalankan kedua algoritma akan didapatkan apakah dengan menggunakan algoritma pemrograman dinamis, algoritma bilangan Fibonacci akan jauh lebih cepat dalam sisi performansi.

Kata kunci: Bilangan Fibonacci. Rekursif, pemrograman dinamis, waktu.

1. PENDAHULUAN

1. Bilangan Fibonacci

Bilangan Fibonacci adalah bilangan dari deret bilangan berikut 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …, menurut definisi, dua bilangan Fibonacci yang pertama adalah 0 dan 1 (ada beberpa sumber yang menyebutkan 1 dan 1) dan setiap bilangan berikutnya adlaah jumalh dari dua buah bilangan sebelumnya, jadi bilangan Fibonacci keempat adalah penjumlahan dari bilangan Fibonacci kedua dan ketiga yaitu satu ditambah dua yaitu tiga.

Dalam rumus matematika deret bilangan Fibonacci dituliskan dalam relasi rekurens berikut ini

$$F_n = F_{n-1} + F_{n-2}$$

Dengan nilai awal

$$F_0 = 0 \text{ dan } F_1 = 1$$

Deret bilangan Fibonacci dinamai sesuai penemunya yaitu Leonardo of Pisa, yang juda dikenal sebagai Fibonacci. Berikut ini beberapa contoh bilangan Fibonacci

F_0=	0	F_1=	1
F_5=	5	F_6=	8
F_{10}=	55	F_{11}=	89
F_{15}=	610	F_{16}=	987
F_{20}=	6,765	F_{21}=	10,946
F_{25}=	75,025	F_{26}=	121,393
F_{30}=	832,040	F_{31}=	1,346,269
F_{35}=	9,227,465	F_{36}=	14,930,352
F_{40}=	102,334,155	F_{41}=	165,580,141
F_{45}=	1,134,903,170	F_{46}=	1,836,311,903

F_2=	1	F_3=	2
F_7=	13	F_8=	21
F_{12}=	144	F_{13}=	233
F_{17}=	1,597	F_{18}=	2,584
F_{22}=	17,711	F_{23}=	28,657
F_{27}=	196,418	F_{28}=	317,811
F_{32}=	2,178,309	F_{33}=	3,524,578
F_{37}=	24,157,817	F_{38}=	39,088,169
F_{42}=	267,914,296	F_{43}=	433,494,437
F_{47}=	2,971,215,073	F_{48}=	4,807,526,976

1.2. Program Dinamis

Program Dinamis (*dynamic programming*) adalah sebuah metode pemecahan maslah dengan menguraikan solusi menjadi sekumpulan langkah

(*step*) atau tahapan (*stage*). Hal ini dilakukan sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Pada penyelesaian persoalan dengan metode ini :

1. terdapat sejumlah berhingga pilihan yang mungkin
2. solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya.
3. kita menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

2. ALGORITMA BILANGAN FIBONACCI

Algoritma dibuat sesuai dengan spesifikasi bilangan dibonacci yang telah disebutkan sebelumnya. Dengan menggunakan rumus $F_n = F_{n-1} + F_{n-2}$ dan nilai awal $F_0 = 0$ dan $F_1 = 1$ maka dapat disusun beberapa algoritma untuk menghasilkan bilangan Fibonacci.

2.1. Menggunakan Metode Rekursif

Dengan metode rekursif kita dapat menyusun algoritma bilangan Fibonacci sebagai berikut :

```
function fib(n)
    if n = 0 return 0
    if n = 1 return 1
    return fib(n - 1) + fib(n - 2)
```

Algoritma diatas menerima input berupa urutan suatu bilangan pada deret bilangan Fibonacci. Jika input yang dimasukkan adalah 0 atau bilangan Fibonacci ke-0 maka akan langsung mengembalikan nilai 0, jika input yang dimasukkan adalah 1 maka akan langsung mengembalikan nilai 1. Untuk nput selain kedua nilai 0 dan 1, maka fungsi diatas akan mengembalikan bilangan Fibonacci ke-(input - 1) ditambah bilangan Fibonacci ke-(input – 2), di bagian inilah metoda rekursif digunakan yaitu dengan memanggil fungsi yang merupakan dirinya sendiri.

Dengan mencoba menelusuri algoritma diatas dengan contoh yaitu fib(5) maka akan dilakukan hal-hal berikut

1. fib(5)
2. fib(4) + fib(3)
3. (fib(3) + fib(2)) + (fib(2) + fib(1))
4. ((fib(2) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1))
5. (((fib(1) + fib(0)) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1))

Jika kita lihat secara teliti, fib(2) dihitung sebanyak tiga kali. Dalam contoh yang lebih besar akan ada banyak nilai dari fungsi fib yang akan dihitung ulang beberapa kali dalam proses rekursif. Hal ini akan membuat program yang menggunakan algoritma ini akan mengkomsumsi *cost* yang sangat banyak jika menggunakan input yang makin besar.

2.2. Menggunakan Metode Rekursif + Program Dinamis

Misal terdapat sebuah object, m, yang bertipe *map*. Object ini akan menyimpan setiap nilai fib yang telah dihitung hasilnya sebelumnya. Dengan memodifikasi algoritma diatas dengan menggunakan object ini maka akan menghasilkan fungsi yang memiliki tingkat kompleksitas O(n) debandingkan eksponensial seperti algoritma sebelumnya. Berikut ini algoritma yang telah diperbaiki :

```
var m := map(0 → 0, 1 → 1)
Function fib(n)
   if map m does not contain key n
     m[n]:= fib(n - 1) + fib(n - 2)
   return m[n]
```

Cara kerja algoritma diatas adalah sebagai berikut: pertama membentuk sebuah object *map* dengan *key* yang digunakan merupakan nilai urutan bilangan Fibonacci, setelah membuat object m maka perlu menginisiasi nilai object m dengan mengisi nilai 0 untuk *key* 0 dan 1 untuk *key* 1, hal ini dilakukan karena kedua nilai tersebut merupakan *seed value* sehingga kita dapat menghitung nilai bilangan dibonacci yang lainnya sesuai dengan aturan deret bilangan Fibonacci.

Setelah menginisiasi object m maka fungsi fib dapat dijalankan. Fungis ini akan melakukan pengecekan terlebih dahulu terhadap input yang dimasukkan apakah telah ada di dalam m apa tidak. Jika sudah ada maka akan langsung mengembalikan nilai yang disimpan pada object m, namun jika object m ternyata belum menyimpan nilai yang diinginkan, maka kan dilakukan proses rekursif dan pada nilai Fibonacci yang didapat akan dimaskukkan ke dalam object m.

Dengan prinsip kerja diatas akan meningkatkan performa dari perhitungan nilai bilangan Fibonacci karena tidak perlu melakukan perhitungan ulang untuk suatu bilangan Fibonacci lebih dari satu kali. Algoritma diatas menggunakan pendekatan secara *top-down.* Karena pertama kita memecah permasalahan menjadi bebeerapa submasalah dan menghitung permasalahn tersebut beserta menyimpan hasilnya.

Teknik yang digunakan pada algoritma diatas disebut *memorization.* Hal ini merupakan sebuah teknik optimasi yang digunakan untuk meningkatkan kecepatan suatu program computer dengan mencegah fungsi melakukan perhitungan secara berulang untuk input yang sama.

Dalam pendekatan secara *bottom-up* kita akan menghitung nilai fib dari input yang lebih kecil terlebih dahulu, lalu menghitung nilai yang lebih besar. Metode ini juga memiliki kompleksitas waktu sebesar O(n)

selama metode ini melakukan *loop* sebanyak n-1 kali. Namun metode ini hanya membutuhkan *space* atau penyimpanan yang konstan (O(1)) untuk menyimpan data, hal ini berkebalikan dengan pendekatan *top-down* yang membutuhkan *space* O(n) untuk menyimpan object bertipe *map*. Berikut ini algoritmanya :

```
function fib(n)
  var previousFib := 0
  var currentFib := 1
   if n = 0
      return 0
   else if n = 1
      return 1
   repeat n - 1 times
      var newFib := previousFib +
currentFib
      previousFib := currentFib
      currentFib  := newFib
   return currentFib
```

cara kerja lagoritma diatas adalah pertama melakukan inisialisasi terhadap previousFib dan currentFib dengan nilai *seed* deret bilangan Fibonacci. Jika fungsi menerima input 0 maka kana langsung mengembalikan nilai 0 jika input yang dimasukkan bernilai 1 maka akan mengembalikan nilai 1. Untuk nilai-nilai input yang lain, maka akan dibuat object baru bernama newFib yang bernilai dari hasil pertambahan previousFib dan currentFib hal ini mengikuti rumus yang talh disebutkan sebelumnya yaitu

$F_n = F_{n-1} + F_{n-2}$

Setelah itu nilai previousFib diubah nilainya menjadi nilai currentFib dan currentFib menjadi newFib. Hal ini dilakukan untuk perhitungan bilangan Fibonacci berikutnya.

Dari kedua lagoritma diatas, sebgai contoh kita hanya akan menghitung nilai fib(2) sebanyak satu kali, dan menggunakannya untuk menghitung fib(3) dan fib(4). Dibandingkan menghitungnya jika salah satunya dihitung.

3. PENGUJIAN

Dengan melihat algoritma-algoritma diatas dapat kita perkirakan bahwa dengan menggunakan program dinamis, perhitungan bilangan Fibonacci akan mampu meningkatkan waktu eksekusi program yang dibuat. Namun seberapa besarkah pengaruh yang dihasilkan hanya dapat dibuktikan dengan melakukan pengujian secara nyata, yaitu dengan membuat programnya dan melakukan eksekusi serta menghitung waktu yang dibutuhkan program untuk menyelesaikan perhitungan.

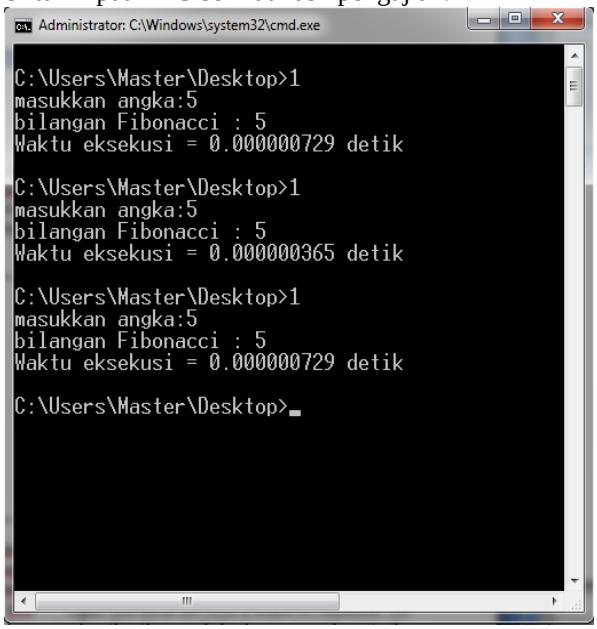
Pengujian dilakukan dengan mencoba mengeksekusi program dan melekukan perhitungan bilangan Fibonacci yang berbeda-beda. Untuk setiap dilakukan tiga kali pengujian untuk setiap algoritma dan dilakukan perhitungan waktu rata-rata untuk mendapatkan waktu yang dibutuhkan untuk menyelesaikan perhitungan.

Program dibuat menggunakan bahasa C dengan compiler GCC dari mingw. Untuk nilai input yang digunakan pada pengujian kali ini adalah nilai 5, 25, dan 45. Hal ini dilakukan untuk melihat perubahan waktu yang dibutuhkan untuk menyelesaikan perhitungan terhadap nilai input.

3.1. Metode Rekursif

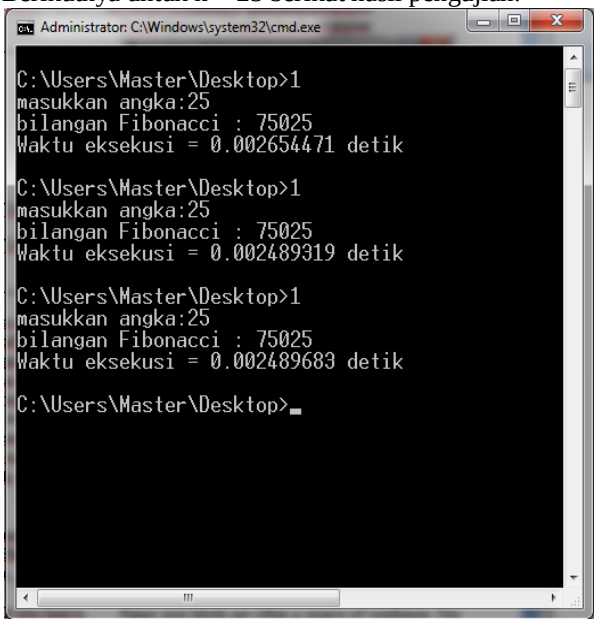
Dengan membuat program sesuai algoritma yang telah diberikan sebelumnya dilakukan pengujian

Untuk input n = 5 berikut hasil pengujian:



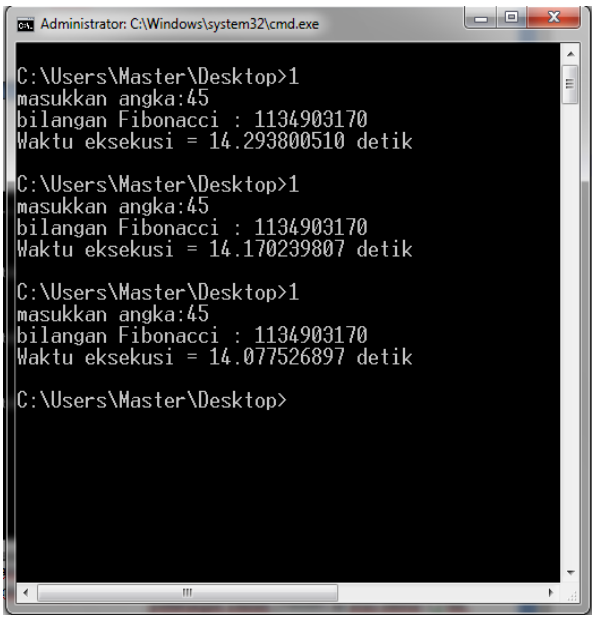
Untuk input n = 5 dapat diketahui bahwa waktu rata-rata yang dibutuhkan untuk program untuk menyelesaikan perhitungan adalah 607 ns

Berikutnya untuk n = 25 berikut hasil pengujian:



Untuk input n = 25 dapat diketahui bahwa waktu rata-rata yang dibutuhkan untuk program untuk menyelesaikan perhitungan adalah 2544491 ns atau sekitar 2,5 ms.

Berikutnya untuk n = 45 berikut hasil pengujian



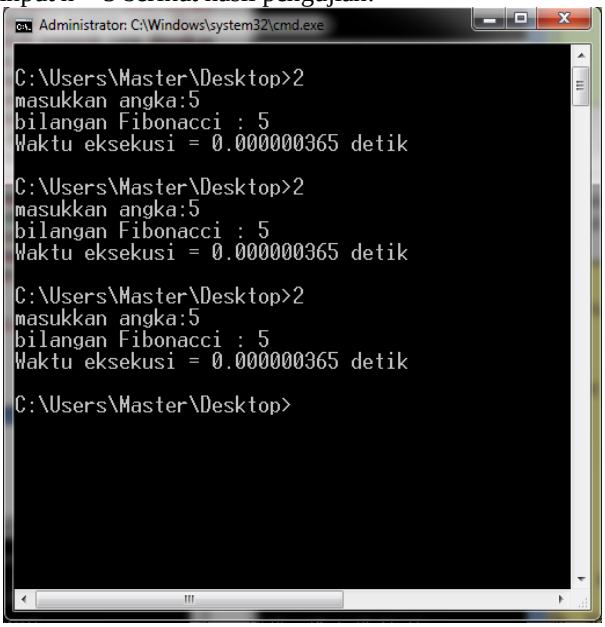
Untuk input n = 45 dapat diketahui bahwa waktu rata-rata yang dibutuhkan untuk program untuk menyelesaikan perhitungan adalah 14,180 s.

3.2. Metode Rekursif + Program Dinamis

3.2.1. Pendekatan Top Down

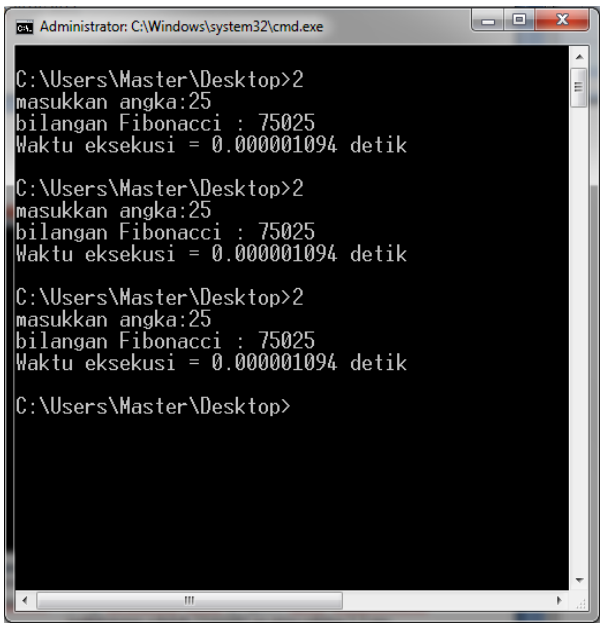
Dengan membuat program sesuai algoritma yang telah diberikan sebelumnya dilakukan pengujian

input n = 5 berikut hasil pengujian:



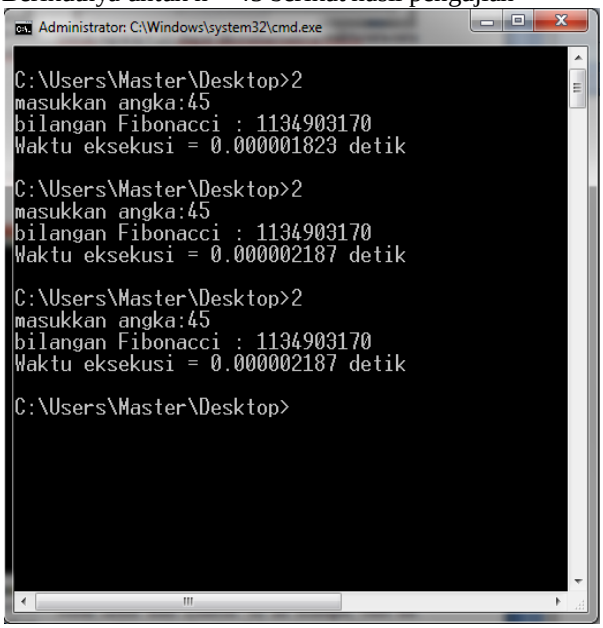
Untuk input n = 5 dapat diketahui bahwa waktu rata-rata yang dibutuhkan untuk program untuk menyelesaikan perhitungan adalah 365 ns

Berikutnya untuk n = 25 berikut hasil pengujian:



Untuk input n = 25 dapat diketahui bahwa waktu rata-rata yang dibutuhkan untuk program untuk menyelesaikan perhitungan adalah 1094 ns.

Berikutnya untuk n = 45 berikut hasil pengujian

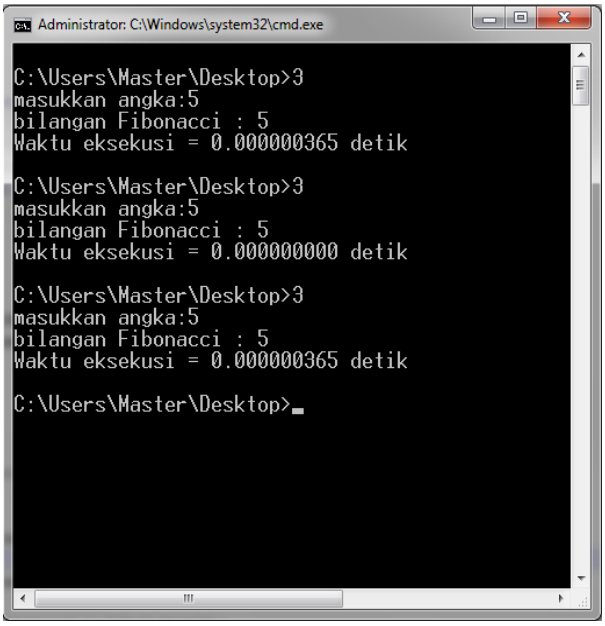


Untuk input n = 45 dapat diketahui bahwa waktu rata-rata yang dibutuhkan untuk program untuk menyelesaikan perhitungan adalah 2065 ns.

3.2.2. Pendekatan Bottom Up

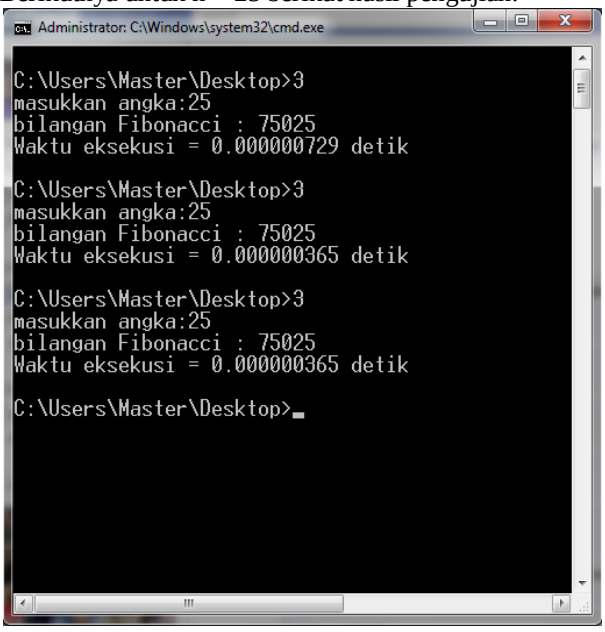
Dengan membuat program sesuai algoritma yang telah diberikan sebelumnya dilakukan pengujian

input n = 5 berikut hasil pengujian:



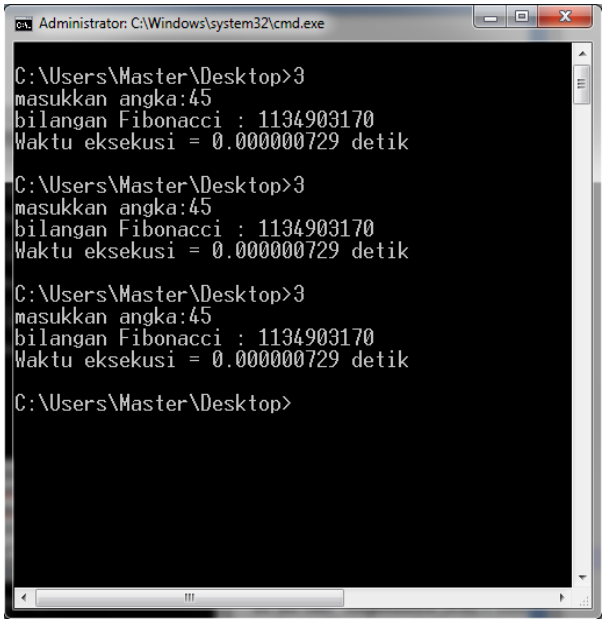
Untuk input n = 5 dapat diketahui bahwa waktu rata-rata yang dibutuhkan untuk program untuk menyelesaikan perhitungan adalah 243 ns

Berikutnya untuk n = 25 berikut hasil pengujian:



Untuk input n = 25 dapat diketahui bahwa waktu rata-rata yang dibutuhkan untuk program untuk menyelesaikan perhitungan adalah 607 ns.

Berikutnya untuk n = 45 berikut hasil pengujian



Untuk input n = 45 dapat diketahui bahwa waktu rata-rata yang dibutuhkan untuk program untuk menyelesaikan perhitungan adalah 729 ns.

4. KESIMPULAN

Dengan melihat hasil pengujian terhadap tiga algoritma yang telah disebutkan, kita dapat melihat bahwa dengan algoritma rekursif + program dinamis akan menghasilkan wwaktu perhitungan yang lebih sedikit atau lebih cepat.

Namun dalam penggunaan program dinamis untuk melakukan perhitungan bilangan Fibonacci ternyata dengan pendekatan *bottom up* didapat waktu eksekusi yang lebih sedikit atau lebih cepat dibandingkan dengan pendekatan *top down*. Padahal jika kedua algoritma tersebut dilihat kompleksitasnya dapat disimpulkan bahwa komleksitas yang dimiliki sama yaitu O(n). hal ini dapat disebabkan oleh kemampuan computer untuk melakukan perhitungan secara rekursif dan traversal.

Komputer akan menyelesaikan perhitungan dengan metode traversal lebih cepat daripada menggunakan rekursif, karena pada proses rekursif, prosesor bekerja lebih berat dengan adanya context switching. Oleh sebab itu pendekatan *bottom up* lebih cepat.

Jadi dapat disimpulkan bahwa penggunaan program dinamis dalam perhitungan bilangan Fibonacci mampu meningkatkan performansi perhitungan.

REFERENSI

http://en.wikipedia.org/wiki/Dynamic_programming. Tanggal akses : 10 Desember 2010 pukul 21:00

http://en.wikipedia.org/wiki/Fibonacci_number. Tanggal akses : 10 Desember 2010 pukul 21:30

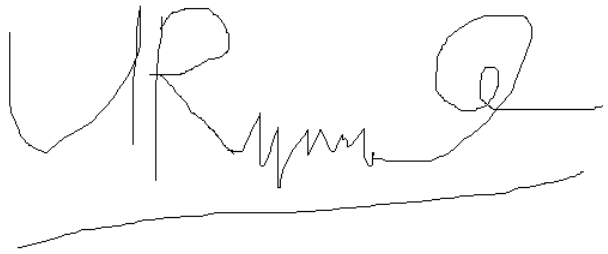
Munir,Rinaldi. “Diktat Kuliah IF3051 Strategi Algoritma”. Institut Teknologi Bandung. 2009

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2010

ttd



Yudi Retanto
13508085