Analisis Kompleksitas Kalkulasi Operasi Pangkat

Mochammad Dikra Prasetya - 13511030
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
dikra.prasetya@students.itb.ac.id

***Abstrak*—Operasi pangkat merupakan hal yang sering ditemui dalam penyelesaian masalah komputasi. Penghitungannya dapat dilakukan dengan beberapa cara. Ada yang berkompleksitas linear sampai yang berkompleksitas logaritmik. Namun setiap cara penyelesaian tersebut memiliki beberapa kelebihan dan kekurangan. Makalah ini akan membahas beberapa algoritma operasi pangkat positif yang sering digunakan pada penyelesaian masalah komputasi dalam pemrograman.**

***Kata kunci—Divide and Conquer*, Kompleksitas, Logaritma, Operasi Aritmatika, Pangkat.**

I. PENDAHULUAN

Operasi aritmatika merupakan salah satu hal yang hamper selalu ditemui dalam penyelesaian masalah pemrograman. Selain operasi dasar '+', '-', '*', '/' yang telah disediakan bahasa pemrograman pada umumnya, operasi pangkat adalah salah satu operasi yang umum untuk digunakan. Contoh dari masalah yang membutuhkan operasi pangkat antara lain masalah polinomial, deret aritmatika, masalah *BigMod*, dan masih banyak lagi.

Pada umumnya soal-soal yang membutuhkan operasi perpangkatan akan berhubungan dengan aritmatika modulo. Hal ini dikarenakan hasil dari perpangkatan sangat besar. Ukuran suatu variabel umumnya sangat terbatas. Ambil contoh pada bahasa-C, *unsigned integer* hanya mampu menampung sampai $2^{64} - 1$.

Pada pembahasan kali ini, diasumsikan tidak ada batasan variabel. Yang akan diperhatikan hanyalah kompleksitas waktu. Cara-cara yang ada pun cukup beragam, dari yang berkompleksitas linear sampai yang berkompleksitas logaritmik. Dengan memanfaatkan teori yang ada, solusi yang efisien pun dapat dicari.

II. DASAR TEORI

A. Operasi Pangkat

Operasi pangkat merupakan operasi yang umum pada persoalan aritmatika. Notasi pangkat secara umum adalah $a^p = a \times a \times .. \times a$ (sebanyak p kali). Operasi ini memiliki beberapa sifat, antara lain :

- $a^p \times a^q = a^{p+q}$
- $a^p \div a^q = a^{p-q}$
- $(a^p)^q = a^{pq}$
- $(a \times b)^p = a^p \times b^p$
- $\left(\frac{a}{b}\right)^p = \frac{a^p}{b^p}, b \neq 0$
- $a^{-p} = \frac{1}{a^p}, a \neq 0$

B. Kompleksitas

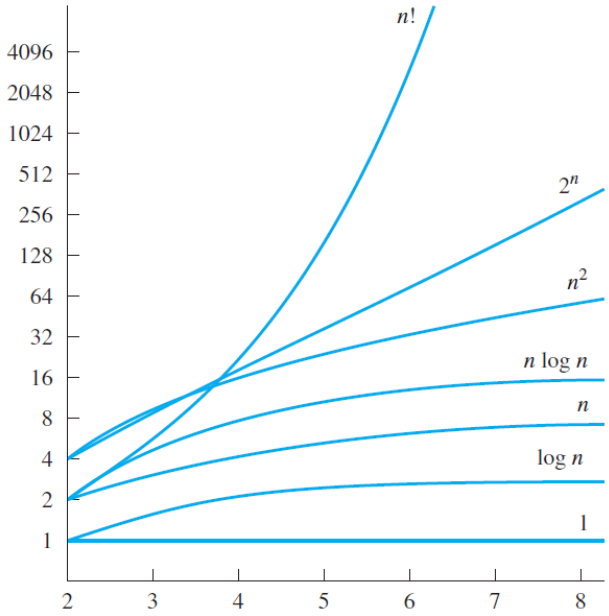
Kompleksitas algoritma memiliki dua macam, yaitu kompleksitas waktu dan kompleksitas memori. Pada makalah ini, kompleksitas yang dibahas adalah kompleksitas waktu.

Kompleksitas waktu, T(n), diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n. Dari notasi T(n) tersebut diambilah limit batas atas dari fungsi tersebut, yang dimana batasan tersebut disebut notasi "O" (Big-O). Notasi tersebut digunakan untuk menyatakan kompleksitas *worst-case* atau biasa disebut notasi asimptotik.

Sebagai batas atas dari kompleksitas waktu, notasi Big-O secara umum digunakan sebagai ukuran efisiensi dari suatu algoritma. Berikut adalah tabel serta grafik dari perkembangan fungsi Big-O berdasarkan ukuran input :

Complexity	*Terminology*
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	Linearithmic complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

Tabel 2.1 Kompleksitas dan terminologinya



Grafik 2.1 Perkembangan fungsi waktu terhadap ukuran input

C. Divide and Conquer

Dalam ilmu komputer, algoritma *Divide and Conquer* merupakan algoritma yang sangat populer. Prinsip dari algoritma ini adalah memecah-mecah masalah yang ada menjadi beberapa bagian kecil sehingga lebih mudah untuk diselesaikan. Langkah-langkah penerapan *Divide and Conquer* pada suatu masalah antara lain :

- *Divide*, bagi masalah yang ada menjadi beberapa sub-masalah
- *Conquer*, selesaikan sub-masalah yang ada secara rekursif
- *Combine*, gabung solusi dari sub-masalah yang ada menjadi solusi dari masalah utama

Contoh dari penerapan *Divide and Conquer* antara lain pada algoritma *Merge Sort*, dengan langkah antara lain berikut :

- *Divide*, bagi *n* elemen yang ada menjadi dua bagian, dengan masing-masing *n/2* elemen
- *Conquer*, urutkan dua bagian yang ada secara rekursif menggunakan *Merge Sort*
- *Combine*, gabung dua bagian yang telah terurut sebelumnya menjadi satu bagian yang terurut.

Rekursi tersebut akan berhenti ketika mencapai bagian dengan elemen 1, dimana bagian ini tidak perlu diurutkan. Bagian ber-elemen 1 tidak perlu diurutkan karena bagian tersebut otomatis sudah terurut (hanya terdiri dari elemen tersebut saja).

D. Operasi Logaritma

Logaritma adalah operasi matematika yang merupakan kebalikan dari eksponen atau pemangkatan. Notasi logaritma pada umumnya adalah $\log_b a = c$,untuk $b^c = a$. Operasi logaritma memiliki beberapa sifat, antara lain :

- $\log(ab) = \log(a) + \log(b)$
- $\log(a/b) = \log(a) - \log(b)$
- $\log(a^b) = b \log(a)$
- $\log\left(\sqrt[b]{a}\right) = \frac{\log a}{b}$
- $\log 1 = 0$
- $\log_b a = \frac{\log a}{\log b}$
- $\log_p p = 1$
- $\log_p(a) \times \log_a(b) = \log_p b$
- $p^{\log_p a} = a$

III. ANALISIS ALGORITMA

A. Solusi Secara Linear

Solusi ini adalah solusi yang paling *straightforward*, atau dengan kata lain solusi yang langsung diturunkan dari definisi operasi pangkat. Sesuai definisi, hasil suatu pangkat adalah perkalian suatu bilangan sebanyak jumlah pangkatnya. Berikut adalah contoh potongan program bahasa-C dari solusi ini :

```
int Pangkat1(int a, int p){
    int hasil = 1;
    for (int i = 0; i < p; ++i){
        hasil *= a;
    }
    return hasil;
}
```

Gambar 3.1 Potongan kode solusi secara linear

Fungsi tersebut akan menghitung hasil operasi pangkat a^p, dengan $p \geq 0$. Variabel hasil akan menampung perkalian a sebanyak p kali. Secara komplekitas waktu, potongan program ini memiliki kompleksitas T(n) = n + 2, dengan kata lain O(n).

Algoritma ini dapat dibuat secara rekursif dengan rumus f(*a*, *p*) sebagai :

$$f(a,p) = \begin{cases} a * f(a,p-1), & p > 0 \\ 1, & p = 0 \end{cases}$$

Berikut potongan program bahasa-C dari fungsi pangkat secara rekursif tersebut :

```
int Pangkat2(int a, int p){
    return (p == 0)? 1 : (a * Pangkat2(a, p-1));
}
```

Gambar 3.2 Potongan kode solusi secara liner dalam bentuk rekursif

Program ini memiliki kompleksitas waktu yang sama dengan cara sebelumnya, yaitu O(n). Maka solusi secara linear dalam bentuk iteratif maupun rekursif relatif tidak akan terlalu berbeda.

B. Solusi Divide and Conquer

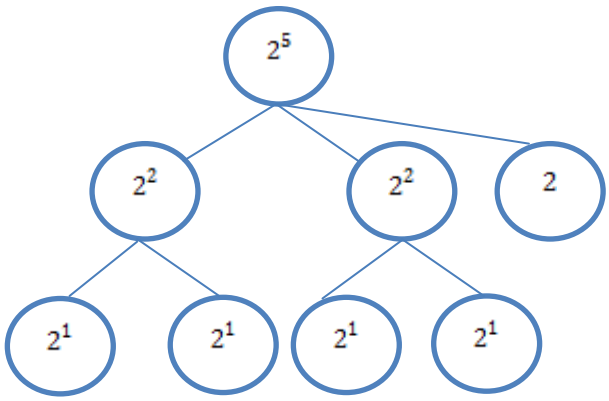
Konsep *Divide and Conquer* ini dapat diterapkan pada operasi perpangkatan. Dengan memanfaatkan sifat bilangan pangkat dimana $a^{p+q} = a^p \times a^q$, maka dapat dibuat rumusan :

$$a^p = \begin{cases} a^{\frac{p}{2}} * a^{\frac{p}{2}}, & p\ genap \\ a^{\frac{p}{2}} * a^{\frac{p}{2}} * a, & p\ ganjil \end{cases}$$

Dengan rumusan tersebut, suatu operasi perpangkatan dapat dirumsukan dalam bentuk *Divide and Conquer*. Langkah-langkah penyelesaiannya antara lain :

- *Divide*, bagi a^p pecahan dua buah $a^{p/2}$
- *Conquer*, selesaikan nilai dari pecahan $a^{p/2}$ secara rekursif
- *Combine*, gabung dua buah pecahan $a^{p/2}$ menjadi satu bagian yaitu a^p, dengan apabila *p* adalah bilangan ganjil maka nilai tersebut dikali *a*

Rekursi tersebut akan berhenti ketika mencapai persoalan a^0 (p = 0) yang bernilai 1, atau a^1 (p = 1) yang bernilai a . Bila diambil contoh a = 2 dan p = 5, maka pohon rekursi tersebut adalah :



Gambar 3.3 Pohon rekursi perpangkatan

Potongan program bahasa-C dari solusi tersebut antara lain :

```
int Pangkat3(int a, int p){
    if (p == 0)
        return 1;

    if (p == 1)
        return a;

    if (p % 2)
        return Pangkat3(a, p/2) * Pangkat3(a, p/2) * a;
    else
        return Pangkat3(a, p/2) * Pangkat3(a, p/2);
}
```

Gambar 3.4 Potongan kode solusi *Divide and Conquer*

Secara kompleksitas waktu, program tersebut memiliki kompleksitas:

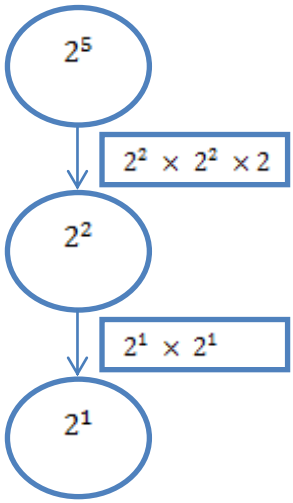
$$T(N) = \begin{cases} 2 * T(\frac{N}{2}), & N > 1 \\ 1, & N \leq 1 \end{cases}$$

Dimana persamaan tersebut menghasilkan :

$$T(N) = 2 \log_2 N \approx 2 \log N$$

Maka dengan T(N) tersebut, kompleksitas waktu dalam notasi Big-O nya adalah O(log N).

Solusi tersebut masih bisa di-optimisasi. Pada solusi tersebut, dapat dilihat perhitungan $a^{p/2}$ dihitung sebanyak 2 kali. Hal ini mengakibatkan suatu perhitungan yang boros, dimana seharusnya perhitungan $a^{p/2}$ dapat dilakukan hanya sekali lalu disimpan pada variabel temporer. Dengan melakukan hal tersebut, pemanggilan fungsi pangkat dapat hanya dilakukan sekali saja. Hal ini mengakibatkan pemanggilan fungsi menjadi dihemat menjadi seperti pohon rekursi berikut ini :



Gambar 3.5 Pohon rekursi perpangkatan

Berikut adalah potongan program-C dari optimisasi diatas :

```
int Pangkat4(int a, int p){
    if (p == 0)
        return 1;

    if (p == 1)
        return a;

    int temp = Pangkat4(a, p/2);

    return temp * temp * ((p % 2)? a : 1);
}
```

Gambar 3.5 Potongan kode optimisasi solusi *Divide and Conquer*

Dengan cara tersebut maka kompleksitas waktunya :

$$T(N) = \begin{cases} 1 + T(\frac{N}{2}), & N > 1 \\ 1, & N \leq 1 \end{cases}$$

Dimana persamaan tersebut menghasilkan :

$$T(N) = \log_2 N \approx \log N$$

Maka didapat notasi Big-O nya akan sama dengan sebelumnya, yaitu O(log N). Walaupun notasi Big-O nya tidak berbeda, namun T(N) nya memiliki perbedaan secara konstanta. Konstanta tersebut tidak akan berpengaruh besar pada input kecil, namun efeknya akan terasa pada input besar. Perbedaan milisekon pun akan berharga untuk menghasilkan program yang lebih optimal.

C. Solusi menggunakan Operasi Logaritma

Solusi ini memanfaatkan operasi logaritma, berikut penurunan rumus nya :

$\rightarrow a^p = c$
$\rightarrow$
$\rightarrow \ln(a^p) = \ln c$
$\rightarrow p \ln a = \ln c$
$\rightarrow \exp(p \ln a) = c$
$\rightarrow a^p = \exp(p \ln a)$

Maka didapatkan hasil suatu perpangkatan a^p adalah exp(p ln a), dengan exp adalah fungsi anti-log (eksponensial). Berikut potongan program bahasa-C dari solusi tersebut :

```
float Pangkat5(int a, int p){
    return exp(p * log(a));
}
```

Gambar 3.6 Potongan kode dengan solusi logaritma

Secara kasat mata, kompleksitas waktu dari program tersebut adalah T(N) = 1, atau dalam notasi Big-O nya O(1). Namun sesungguhnya operasi fungsi *exp* dan fungsi *log* pada program tersebut memiliki kompleksitas waktu yang tidak tentu. Operasi fungsi yang disediakan oleh *library* pendukung matematika pada setiap bahasa pemrograman pada umumnya memiliki kompleksitas yang mungkin membesar secara eksponensial terhadap ukuran input. Sifat ini mengakibatkan cara ini merupakan solusi yang tidak stabil, sehingga tidak umum untuk digunakan.

V. KESIMPULAN

Terdapat beberapa cara yang dapat digunakan untuk membuat fungsi pangkat pada pemrograman. Diantara cara-cara yang telah dibahas, solusi dengan penerapan *Divide and Conquer* merupakan solusi yang paling efektif dan aman digunakan.

DAFTAR PUSTAKA

[1] Rosen, Kenneth H. "Discrete Mathematics and Its Applications Sixth Edition." 2007. McGraw-Hill.
[2] http://telkomselsahabatguru.com/browse/materi/Kompetensi%20I/Bentuk%20Akar/new_page_3.htm 4.17PM 17/12/2012
[3] http://id.wikipedia.org/wiki/Divide_and_Conquer 9.23PM 17/12/2012
[4] http://id.wikipedia.org/wiki/Logaritma 9.28PM 17/12/2012
[5] http://www.catonmat.net/blog/mit-introduction-to-algorithms-part-two/ 10.03PM 17/12/2012
[6] http://www.rumus.web.id/matematika/sifat-logaritma-matematika/ 11.24PM 18/12/2012
[7] http://en.wikipedia.org/wiki/Big_O_notation 11 44PM 18/12/2012

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, 17 Desember 2012

Mochammad Dikra Prasetya / 13511030