Michael Johanes Johansyah
L0123081
Kelas C

TUGAS PSDA 02 & 03

Setelah memahami efisiensi algoritma, saya dapat memilih dan merancang algoritma yang sesuai dengan kebutuhan untuk menyelesaikan masalah. Adapun berikut ini beberapa contoh algoritma rekursif beserta hasil analisis kompleksitas dari masing-masing algoritma:

1. Algoritma Faktorial

```
#include <iostream>

int main(){
int factorial(int n); // Deklarasi fungsi

int result = factorial(5); // Menampung nilai kembalian

printf("Faktorial 5 adalah %d\n", result);

return 0;
}

int factorial(int n) {
  // Kasus dasar: Faktorial 0 adalah 1
  if (n == 0) {
    return 1;
  }
  // Rekursi: Faktorial n adalah n dikali dengan faktorial (n-1)
  else {
    return n * factorial(n - 1);
  }
}
```

Dengan menggunakan algoritma ini, hasil analisis kompleksitas yang saya dapatkan adalah algoritma ini memiliki notasi Big O : O(n!). Karena semakin tinggi nilai n, semakin kompleks pula komputer menghitung hasil jawabannya, hal ini dapat menyebabkan komputer tidak mampu lagi menghitung hasil jawabannya

Adapun saya menemukan error pada kode ini, dimana hasil jawaban hanya akan akurat hingga faktorial ke 12, saat masuk ke faktorial ke 13, angka yang diberikan tidak sesuai jawaban sebenarnya. Dari sini saya dapat menyimpulkan bahwa algoritma faktorial adalah algoritma yang sangat tinggi tingkat kompleksitasnya, sehingga dapat langsung menghabiskan memori suatu komputer.

```
Faktorialnya adalah 1932053504
```

Hasil dari faktorial 13, seharusnya jawabannya adalah 6.227.020.800.

2. Algoritma Perpangkatan Dua (2^n)

```c
#include <iostream>

int main(){
int pow2(int n); // Deklarasi fungsi

int result = pow2(5); // Menampung nilai kembalian

printf("Hasil perpangkatan duanya adalah %d\n", result);

return 0;
}

int pow2(int n) {
  if (n == 0) {
    return 1;
  } else {
    return 2 * pow2(n - 1);
  }
}
```

Dengan menggunakan algoritma ini, hasil analisis kompleksitas yang saya dapatkan adalah algoritma ini memiliki notasi Big O : O(2^n). Karena semakin tinggi nilai n, semakin kompleks pula komputer menghitung hasil jawabannya, hal ini dapat menyebabkan komputer tidak mampu lagi menghitung hasil jawabannya

Adapun saya menemukan error pada kode ini, dimana hasil jawaban hanya akan akurat hingga perpangkatan dua ke 30, saat masuk ke perpangkatan ke 31, angka yang diberikan tidak sesuai jawaban sebenarnya. Dari sini saya dapat menyimpulkan bahwa algoritma perpangkatan dua adalah algoritma yang cukup tinggi tingkat kompleksitasnya, namun tidak setinggi algoritma faktorial karena nilai n disini bisa mencapai 30, tidak seperti faktorial yang hanya 12.

```
Hasil perpangkatan duanya adalah -2147483648
```

Hasil dari 2^31, seharusnya jawabannya adalah 2.147.438.648, hasil diatas menunjukkan angka minus. Sedangkan hasil dari 2^32 menunjukkan angka nol (0).

3. Algoritma Fibonacci

```
#include <iostream>

int main(){
int fibonacci(int n); // Deklarasi fungsi

int result = fibonacci(47); // Menampung nilai kembalian

printf("Hasil Fibonaccinya adalah %d\n", result);

return 0;
}

int fibonacci(int n) {
  if (n == 0 || n == 1) {
    return n;
  } else {
    return fibonacci(n - 1) + fibonacci(n - 2);
  }
}
```

Dengan menggunakan algoritma ini, hasil analisis kompleksitas yang saya dapatkan adalah algoritma ini memiliki notasi Big O : O(2^n), sama dengan perpangkatan dua. Karena semakin tinggi nilai n, semakin kompleks pula komputer menghitung hasil jawabannya, hal ini dapat menyebabkan komputer tidak mampu lagi menghitung hasil jawabannya

Adapun saya menemukan error pada kode ini, dimana hasil jawaban hanya akan akurat hingga perpangkatan dua ke 46, saat masuk ke perpangkatan ke 47, angka yang diberikan tidak sesuai jawaban sebenarnya. Dari sini saya dapat menyimpulkan bahwa algoritma Fibonacci adalah algoritma yang cukup tinggi tingkat kompleksitasnya, namun tidak setinggi algoritma faktorial dan perpangkatan dua karena nilai n disini bisa mencapai 46, tidak seperti faktorial yang hanya 12, dan perpangkatan dua yang hanya 30.

Namun ada hal unik juga yang saya temukan pada algoritma ini. Saat saya mencari angka 40 ke atas, terasa sekali waktu yang dibutuhkan untuk menghasilkan jawaban menjadi lebih lama, hal ini tidak saya alami pada dua algoritma sebelumnya. Hal ini dikarenakan pada setiap pemanggilan rekursi, n dikurangi dengan 1 atau 2.

Hal ini menyebabkan "ledakan" dalam jumlah pemanggilan rekursi, yang meningkat secara eksponensial dengan n. Semakin besar nilai n, semakin banyak pemanggilan rekursi yang dilakukan, dan semakin lama waktu yang dibutuhkan untuk menyelesaikannya.