2020/1 『알고리즘』 과제 보고서					
학번	201418062	이름	레토 아치마드 아판디	제출일자	2020. 04. 07
제목	알고리즘				

Analysis of Algorithm

Analysis of algorithms is the determination of the amount of time and space resources required to execute it. Usually, the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps, known as time complexity, or volume of memory, known as space complexity.

A complete analysis of the running time of an algorithm involves the following steps:

- Implement the algorithm completely.
- Determine the time required for each basic operation.
- Identify unknown quantities that can be used to describe the frequency of execution of the basic operations.
- Develop a realistic model for the input to the program.
- Analyze the unknown quantities, assuming the modelled input.
- Calculate the total running time by multiplying the time by the frequency for each operation, then adding all the products.

15.Showdirectly that $\mathrm{F(n)} = n^2 + 3n^3 \in \theta(n^3). that\ is, use\ the\ definitions\ of\ O\ and\ \Omega\ to\ show\ that\ f(n) is$ both $\mathrm{O}(n^3)\mathrm{and}\ \Omega(n^3)$.

SOLUTION :

First: we use the definition of Big oh.

$\mathrm{f(n)} \leq \mathrm{cg(n)} for\ all\ n \geq k$
now $\mathrm{f(n)} = n^2 + 3n^3\ and\ g(n) = o(n^3)$

by using the definition of big oh.
$n^2 + 3n^3 \leq c\ n^3$

If the value of n=1 and c=4 then the condition is satisfied.
$n^2 + 3n^3 \leq 4n^3$ // c=4 & n=1
Put value of c and c
$28 \leq 32$
So f(n) is $\mathrm{O}(\mathrm{n}^3)$
Second : we use the definition of Big omega
F(n) $\geq \mathrm{cg(n)} for\ all\ n \geq k$

now f(n) = $n^2 + 3m^3 and\ g(n) =$ $\Omega(n^3)$
$n^2 + 3n^3 \geq c\ n^3$

If the value of n=1 and c=1 then the condition is satisfied.
$n^2 + 3n^3 \geq cn^3$ // c=1 and n=1
Put the value of c and n
4≥ 1
So f(n) is $\Omega(n^3)$

Because we have shown that f(n) is $O(n^3)\ and\ f(n) is\ \Omega\ (n^3)$, it must be that $f(n) is\ \theta(n^3)$

18. Let $p(n) = akn^k + ak - 1nk - 1 + \cdots + a1n + a0, whereak > 0.$ Using the properties of order in section 1.4.2, show that $p(n) \in \theta(n^k)$.

Sol :

I. $p(n) \in O(n^k)$:

$$\begin{aligned} p(n) &= a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0 \\ &= a_k \cdot n^k \cdot \left(1 + \frac{a_{k-1}}{a_k}\frac{1}{n} + \ldots + \frac{a_0}{a_k}\frac{1}{n^k}\right) \\ &\leq a_k \cdot n^k \cdot \left(1 + \frac{|a_{k-1}|}{a_k}\frac{1}{n} + \ldots + \frac{|a_0|}{a_k}\frac{1}{n^k}\right) \\ &\leq a_k \cdot n^k \cdot \left(1 + \frac{|a_{k-1}|}{a_k} + \ldots + \frac{|a_0|}{a_k}\right) \end{aligned}$$

for all $n \geq 1$. With

$$c := a_k \cdot \left(1 + \frac{|a_{k-1}|}{a_k} + \ldots + \frac{|a_0|}{a_k}\right)$$

$p(n) \leq c \cdot n^k$ for all $n \in \mathbb{N}$ holds. Thus $p(n) \in O(n^k)$.

II. $p(n) \in \Omega(n^k)$:

$$\begin{aligned} p(n) &= a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0 \\ &= a_k n^k \cdot \left(1 + \frac{a_{k-1}}{a_k}\frac{1}{n} + \ldots + \frac{a_0}{a_k}\frac{1}{n^k}\right) \end{aligned}$$

The expression within parentheses has limit 1 for $n \to \infty$. Thus, there is $n_0 \in \mathbb{N}$, such that this expression is $\geq 1/2$ for all $n \geq n_0$. Then, for $c := \frac{1}{2}a_k$ und $n \geq n_0$

$$p(n) \geq \frac{1}{2}a_k n^k = c \cdot n^k$$

holds. Thus $p(n) \in \Omega(n^k)$.

III. $p(n) \in \Theta(n^k)$ holds because of I. and II.

22. Group the following functions by complexity category.

$n \ln n$ $\quad (\lg n)^2$ $\quad 5n^2 + 7n$ $\quad n^{5/2}$

$n!$ $\quad 2^{n!}$ $\quad 4^n$ $\quad n^n$ $\quad n^n + \ln n$

$5^{\lg n}$ $\quad \lg(n!)$ $\quad (\lg n)!$ $\quad \sqrt{n}$ $\quad e^n$ $\quad 8n + 12$ $\quad 10^n + n^{20}$

Result :

n^n and $(n^n + \ln n)$
$n!$
$10^n + n^{20}$
4^n
e^n
$(lgn)!$
$n^{5/2}$
$5^{\lg n}$
$5n^2 + 7n$
$n \log n$ and $\log(n!)$
$8n + 12$
$n^{1/2}$
$\log^2 n$

25. Suppose you have a computer that requires 1 minute to solve problem instances of size n = 1000. Suppose you buy a new computer that runs 1,000 times faster than the old one. What instances sizes can be run in 1 minute, assuming the following time complexities T(n) for our algorithm ?
(a) T(n) = n
(b) T(n) = n^3
(c) Y(n) = 10^n

Let's say that T(n) is the time required to perform n operations (op).

We know that T(1000) = 1' in the old machine and T(1000) = 1'/1000 in the new one (because the new one is 1000 faster.)

a) T(n) = n
This means 1000 op in 1' in the old machine. In the new machine (1000 times faster) is 1000 op in 1'/1000. So 1000000 op in 1' in the new machine, and the answer is 1 million. In other words, the new machine will compute 1000000 operations in 1' (this is 1000 times more operations per minute.)

b) T(n) = n^3
Here we have 1000^3 op in 1' (old machine) and 1000^3 op in 1'/1000 (new machine). So, in the new machine: 1000^3 1000 op in 1', or 1000^3 10^3 op in 1', i.e., 10000^3 op in 1', and the answer is ten thousand.

c) T(n) = 10n
We have 10 * 1000 op in 1' (old), 10 * 1000 op in 1'/1000 (new). So, 10 * 1000000 op in 1', and the answer is again 1 million.

As we can see the answers for T(n) = n and T(n) = 10 n are the same. In fact, they would also be equal to the answer for T(n) = C n, no matter the value of C > 0. To see this, we only have to replace 10 with C above:

C 1000 op in 1' (old), C 1000 op in 1'/1000 (new) or C 1000000 in 1', and the answer is 1000000.

This is why we talk about O(n), O(n^2), O(nˆ3), etc., regardless of the constant hidden inside the O.

34. What is the time complexity T(n) of the nested loops below? For simplicity, you may assume that n is a power of 2. That is, n = 2^k for some positive integer k.

```
i = n;
while (i >= 1){
    j = i;
    while (j <= n){
        < body of the while loop>    //Needs Θ(1).
        j = 2 * j;
    }
    i = ⌊i/2⌋;
}
```

The answer is $O(n^2)$