Data Structure Homework 1

Student ID : 41247001S

Question 1 :

Use the definition of big-O to prove that $3n^2$ + $2nlog_2n^2 = O(n^2)$. Provide appropriate constants c and $n0$.

Answer :

According to the definition of Big-O , if $f(x) \leq cn^2$ then $f(x) = O(n^2)$
Therefore, while $3n^2$ + $2nlog_2n^2 \leq cn^2$, $3n^2$ + $2nlog_2n^2 = O(n^2)$

If there is any c satisfied, $3n^2$ + $2nlog_2n^2$ can be proven it is equal to $O(n^2)$

$3n^2$ + $2nlog_2n^2 \leq cn^2$

$= 3n^2$ + $2n * 2log_2n \leq cn^2$

$= 3n^2$ + $4n * log_2n \leq cn^2$

$= 3$ + $(4log_2n)/n \leq c$

Since $(log_2n)/n \leq 1$, 3 + $(4log_2n)/n \leq 7$
Therefore, when $c = 7$, $3n^2$ + $2nlog_2n^2 \leq cn^2$
Any $n0$ which satisfy $n \geq n0$, $(log_2n)/n \leq 1$ can be answer

It proves that $3n^2$ + $2nlog_2n^2 \leq cn^2$ is valid and c , $n0$ = 1.
which means $3n^2$ + $2nlog_2n^2 = O(n^2)$

Question 2 :

Show that $4n^3 + 8n^2 + 2^n$ = $\Omega()$.
Please find the maximum order for the big-Ω estimation.
Please also provide the values of c and n0 satisfying the definition of the big-Ω estimation.

Answer :

According to the definition of Big-Ω ,if $f(x) \geq c * g(x)$, $f(x) = \Omega(g(x))$
Since the question want **maximum order** , we can set the $g(x) = 2^n$
(It's because that 2^n has the maxinum growth speed in $4n^3 + 8n^2 + 2^n$)

$4n^3 + 8n^2 + 2^n \geq 2^n = 4n^3/2^n + 8n^2/2^n + 1 \geq c$
If we set $c = 1$ $n >= 0$, $4n^3/2^n + 8n^2/2^n + 1 \geq 1$ Therefore when $c = 1$, $n0 = 1$

Question 3 :

Please determine a succinct big-Θ expression for the growth of the function
$log(n^2) + n^2log(n^4) + 1000n^3 + 5000000n$.

You don’t have to provide appropriate constants c1, c2, and n0 for the definition.
However, please explain how to get your answer.

Answer :

According to the definition of Big-Θ ,while $c1g(x) \geq f(x) \geq c2g(x)$, $f(x) = \Theta(g(x))$

The professor had told us that we should observe the part which growth fastest in the function
while we want to calculate the time complexity of a function $f(x)$,
The given function $log(n^2) + n^2log(n^4) + 1000n^3 + 5000000n$,
we can see that the fastest growth part is $1000n^3$
Therefore, we can set the $g(x)$ to n^3
Since we don't have to provide appropriate constants $c1, c2, n0$
Therefore we can set the answer to $\Theta(n^3)$,according that n^3 growth fastest in the function.

Question 4 :

Analyze and give the time complexity of the following program segments in terms of n.
Please briefly explain your answer.

4-1 :

```
int value = 0;
for(int i=0;i<n;i++)
    for(int j=0;j<i;j++)
        value += 1;
```

Answer of 4-1

```
 //  code                        Freq            Total Steps
int value = 0;                   1                    1
for(int i=0;i<n;i++)             n+1                  n+1
    for(int j=0;j<i;j++)         i+1                 ((1+n)*n)/2 + 1
        value += 1;              1                   ((1+n)*n)/2
```

Therefore, the time complexity = $((1 + n) * n)/2$ = $O(n^2)$

4-2 :

```
for (int i = 1; i < n; i++) {
    i *= k;
}
```

Answer of 4-2

```
// code                            Freq              Total Steps
for (int i = 1; i < n; i++) {     logn / logk          logn / logk
    i *= k;                         1                  logn / logk
}
```

Therefore, the time complexity = $logn/logk$ = $O(log_k n)$

4-3 :

```
int i, j, k = 0;
for (i = n / 2; i <= n; i++) {
    for (j = 2; j <= n; j = j * 2) {
        k = k + n / 2;
    }
}
```

Answer of 4-3

```
// code                                  Freq           Total Steps
int i, j, k = 0;
for (i = n / 2; i <= n; i++) {           (n/2) + 1        (n/2) + 1
    for (j = 2; j <= n; j = j * 2) {     log(n)           log(n)*(n/2)
+ 1
        k = k + n / 2;                      1             log(n)*(n/2)
    }
}
```

Therefore, the time complexity = $log(n) * (n/2)$ = $O(nlogn)$

Question 5

(1) Why when we implement the time complexity (bigO), we usually ignore the coef of n. (Example : we imply $O(n^2)$ but not $O(5n^2)$)?
(2) If there is a function $5n^3 + 4n + 2$,Why we usually imply its time complexity as $O(n^3)$ but not $O(n^3 + n)$?

Answer

(1) The main reason is when we calculate the time complexity, while the n growth to a very large number, the coef will not affect the time complexity much. For example, if a function's time complexity is $O(n^2)$, it doesn't really matter it is $5n^2$ or n^2 while n is large.

On the other hand, the main goal when we imply the time complexity is to estimate the efficiency of a function, not the precious number.
Therefore, while we are estimating Big-O notation, the important part is the growth speed, not the coef.

(2) The main reason is the growth speed of n^3 is too fast.
Therefore, while n changes, n^3 grow too fast compared to other part of function.
We can ignore the part of the function which grow slower since it doesn't affect the whole time speed a lot.

There is an important knowledge between these two question, which we should focus on the time efficiency but not the precious number of the function when we are estimating the time complexity of an algorithm.