1.

a. $100n + logn \cong n + (logn)^2$

logn is not steep enough compared to n. So we should compare just 100n and n.

f(g(n)) = $\Theta(g(n))$

b. $logn \cong \log(n^2) = 2logn$

f(g(n)) = $\Theta(g(n))$

c. $\frac{n^2}{logn} > n(\log n)^2$

As I said logn is not steep enough compared to n.

$$n^2 > n(\log n)^3$$

Then n square is much bigger.

f(g(n)) = $\Omega\ (g(n))$

d. $n^{\frac{1}{2}} < (logn)^5$

Imagine we put 100 in those functions.

10 < 32

f(g(n)) = O(g(n))

e. $n2^n < 3^n$

$$n < \frac{3^n}{2^n}$$

f(g(n)) = O(g(n))

2.

a. The algorithm is to return the maximum

b. Temp1 = T(n/2) + c => it divides the array into half

Temp2 = T(n/2) + c => it also divides the array into half

Therefore the recurrence relation is **2T(n/2) + cn**

When we draw the Tree, the height is log_2^n

So, the constant is $\boldsymbol{cn}(log_2^n) + \boldsymbol{cn}$

level	Level number	Total # of recursive executions at this level	Input size to each recursive execution	Work done by each recursive execution, excluding the recursive calls	Total work done by the algorithm at this level
Root	0	1	N	cn	cN
One Level below root	1	2	n/2	cn/2	cn
Two Levels Below Root	2	4	n/4	cn/4	cn
The Level Just Above The Base Case Level	$log_2^n - 1$	$2^{(log_2^n - 1)}$	$\frac{n}{2^{log_2^n - 1}}$	$\frac{cn}{2^{log_2^n - 1}}$	cn
Base Case level	log_2^n	$2^{(log_2^n)}$	1	c	cn

c.

$nlog_2^n$

3.

level	Level number	Total # of recursive executions at this level	Input size to each recursive execution	Work done by each recursive execution, excluding the recursive calls	Total work done by the algorithm at this level
Root	0	1	N	Cn	cn
One Level below root	1	7	n/8	cn/8	7cn/8
Two Levels Below Root	2	49	n/64	cn/64	49cn/64
The Level Just Above The Base Case Level	$log_8^n - 1$	$7^{(log_7^n - 1)}$	$\frac{n}{8^{log_7^n - 1}}$	$\frac{cn}{8^{log_7^n - 1}}$	$\frac{7^{log_7^n - 1} * cn}{8^{log_7^n - 1}}$
Base Case level	log_8^n	$7^{(log_7^n)}$	1	c	cn

T(n) = cn(1/(1-7/8) = 8cn

4.

- Base case proof

  T(1) = O(n)

  As T(1) = 5, T(1) = O(n)

- Inductive hypothesis

  T(k) = O(k)

  For some c and N for all n>N i.e. T(k) < 차

- To prove

  T(K+1) = O(K+1)

  Using induction

  T(K+1) < 3CK/3 + 5

  T(K+1) < C(K+1) + 5 -C

  When c = 5,

  ⇨ T(K+1) < C(K+1)

5.

If

S(n) = $n^4 + n^2$

f(n) = n^4

r(n) = $n^4 - n^2$

g(n) = n^4

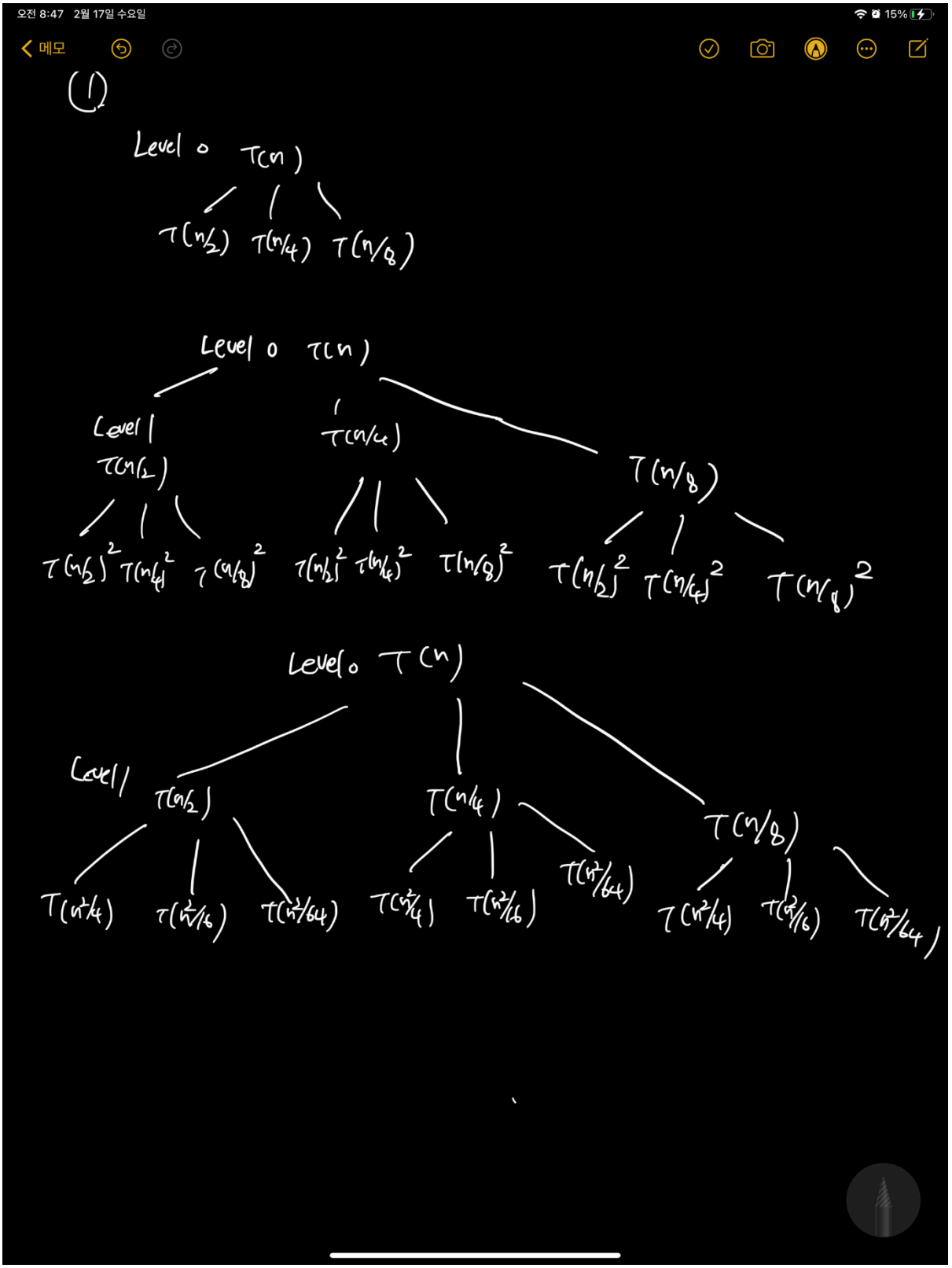
Therefore, f(n) = O(s(n)), g(n) = O(r(n))

⇨ f(n) – g(n) = 0 = O(1)

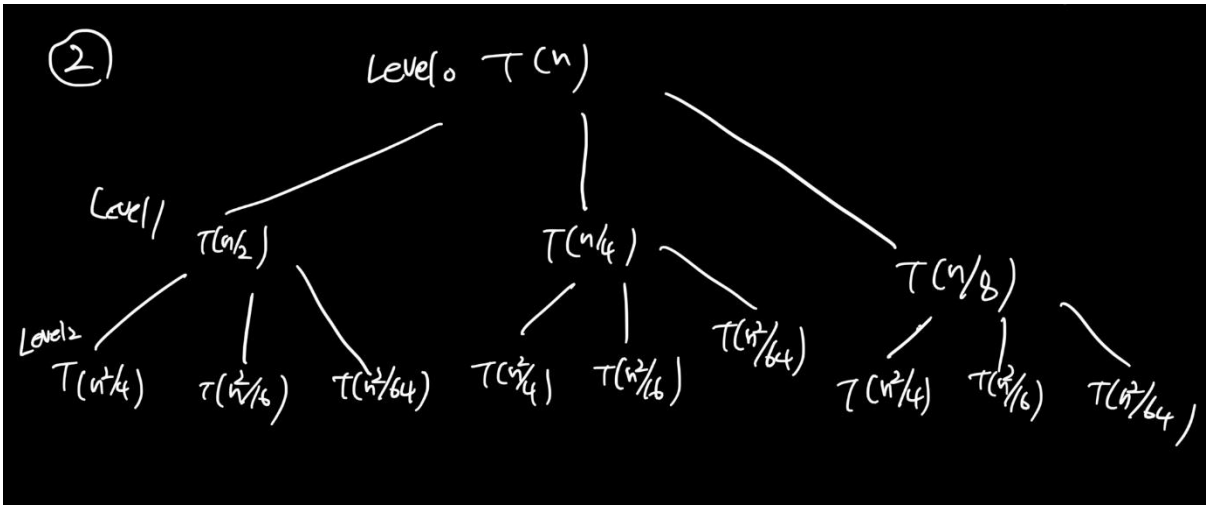
⇨ O(s(n) -r(n)) = O($n^4 + n^2 - (n^4 - n^2)$)

⇨ Therefore, $f(n) - g(n) \neq O(s(n) - r(n))$

6.


오전 8:47 2월 17일 수요일
15%
메모
(1)
Level 0 T(n)
T(n/2) T(n/4) T(n/8)
Level 0 T(n)
Level 1
T(n/2)
T(n/4)
T(n/8)
Level 0 T(n)
Level 1
T(n/2)
T(n/4)
T(n/8)

(2)



(3)

The height k is given by $n(1/2)^k \leq 1$, meaning $n \leq 2^k$ or $k \geq \log_2 n$

(4)

$n < 8^k$ or $k \geq \log_8 n$

(5)

- Assume for all $i < n$ that $c_1 n \leq T(i) \leq c_2 n$.

$\Rightarrow c_1 n/2 + c_1 n/4 + c_1 n/8 + kn \leq T(n) \leq c_2 n/2 + c_2 n/4 + c_2 n/8 + kn$

$\Rightarrow c_1 n\left(\frac{7}{8} + k/c_1\right) \leq T(n) \leq c_2 n\left(\frac{7}{8} + \frac{k}{c_2}\right)$

$\Rightarrow$ If $c_1 \geq 8k$ and $c_2 \leq 8k$, then $c_1 n \leq T(n) \leq c_2 n$. So, $T(n) = \Theta(n)$

7.

Statement of what you have to prove :

We have to show that T(n) = O(n) for T(n) = T(n/2)+ T(n/4) + T(n/8) + T(n/8) + n; T(1) = c.

Base case proof:

Assume that T(n) = c *n for some c which is bigger than 0. Let n =1, c =1. We have T(1) = 1 = c.

Inductive Hypothesis :

T(n) = T(n/2)+ T(n/4) + T(n/8) + T(n/8) + n

= c(n/2) + c(n/4) + c(n/8) + c(n/8) + n

= 7/8 cn+ n =n (7/8 c +1)

Inductive Step :

We now should find a constant that is into formula above.

C = (7/8c + 1) -> c = 8 -> 7/8 (8) + 1 = (7 + 1) = 8. There fore, c = 8 and T(n) = 8c

⇨ T(n) = O(n)

8.

a. T(n) = 2T(99n/100) + 100n

= 198T(n/100) + 100n

Based on the Masters theorem, a = 198, b = 100, c = 1, f(n) = 100n

$$log_b^a = log_{100}^{198}$$

$$log_{100}^{198} > 1 = c$$

⇨ T(n) = $\theta\left(n^{log_b^a}\right) = \theta(n)$

b.

Based on the Masters theorem, a = 16, b = 2, c = 3, k = 1

$$log_b^a = log_2^{16} > 3 = c$$

⇨ T(n) = $\theta\left(n^{log_b^a}\right) = \theta(n^4)$

c.

Based on the Masters theorem, a = 16, b = 4, c = 2

$$log_b^a = log_4^{16} = 2 = c$$

⇨ T(n) = $\theta(n^{2logn})$

9.

(1)

T(n) = 2T(n-1)+1

T(1) = 2T(0)+1 = 3 = $(2^{n+1} - 1)$

T(2) = 2T(1)+1 = 7 = $(2^{n+1} - 1)$

T(3) = 2T(2)+1 = 15 = $(2^{n+1} - 1)$

(2)

T(n) = 2T(n-1) + 1 => 2(2T(n-2)+1)+1

= 4T(n-2) + (1+2)

=4(2T(n-3)+1)+3

=$2^n T(0) + (1 + \cdots + 2^{n-1})$

= $2^{n+1} - 1$

(3)

- Base case

N = 0, T(0) = 1

- Let's assume that T(k) = $2^{k+1} - 1$ for k >0

  T(k+1) = 2T(k)+1 = same.

  Proved.

(4)

Because the order is 2^n, it will have 2^n times operations to show the output.

10.

T(n) = T(n-1) + n/2

= T(n-2) + ((n-1)+n)/2

⇨ T(n-k) + ((n-k+1)+(n-k+2)+…+n)/2

Put k = n – 1

= T(1) + (2+3+…+n)/2

= 1+(1+2+3+…+n-1)/2

= $\frac{n(n+1)}{4} + \frac{1}{2}$

11.

T(n) = 2T(n/2) + $2nlog_2^n$

⇨ 2(2T(n/4) + 2(n/2) $log_2^{n/2}$) + $2nlog_2^n$

⇨ 4T(n/4) + 4n$log_2^n - 2n$

⇨ 8T(n/8) + 6nlog_2^n-6n

⇨ $2^kT\left(\frac{n}{2^k}\right) + 2knlog_2^n - 2kn$

For $\frac{n}{2^k} = 2$, $2^k = n/2$

= n/2 * T(4) + $2(log_2^n - 1)nlog_2^n - 2(log_2^n - 1)n$

= 2n + $2(log_2^n - 1)nlog_2^n - 2(log_2^n - 1)n$

⇨ O($nlog_2^2n$)

12.

O(n^2) is the worst case running time. Therefore, when we use it, we should say “at most” rather than “at least”. Because it implies the upper bound.