Logarithms

We shall use the following notations:

$$\lg n = \log_2 n \quad \text{(binary logarithm)} ,$$
$$\ln n = \log_e n \quad \text{(natural logarithm)} ,$$
$$\lg^k n = (\lg n)^k \quad \text{(exponentiation)} ,$$
$$\lg \lg n = \lg(\lg n) \quad \text{(composition)} .$$

An important notational convention we shall adopt is that *logarithm functions will apply only to the next term in the formula*, so that $\lg n + k$ will mean $(\lg n) + k$ and not $\lg(n + k)$. If we hold $b > 1$ constant, then for $n > 0$, the function $\log_b n$ is strictly increasing.

For all real $a > 0$, $b > 0$, $c > 0$, and n,

(3.14)
$$\begin{aligned} a &= b^{\log_b a} , \\ \log_c(ab) &= \log_c a + \log_c b , \\ \log_b a^n &= n \log_b a , \\ \log_b a &= \frac{\log_c a}{\log_c b} , \end{aligned}$$

(3.15)
$$\begin{aligned} \log_b(1/a) &= -\log_b a , \\ \log_b a &= \frac{1}{\log_a b} , \\ a^{\log_b c} &= c^{\log_b a} , \end{aligned}$$

where, in each equation above, logarithm bases are not 1.

Solutions to exercises

(1) In mathematica (or python) define the functions and compute the number of operations:

e.g. f(n)=Sqrt(n) —> f(n)/10^6 == x Sec —> n = f^-1(x 10^6)

Item	1 second	1 miniute	1 hour	1 day	1 month	1 year	1 century
$\lg n$	2^{10^6}	2^{6*10^7}	2^{36*10^8}	2^{864*10^8}	$2^{25920*10^8}$	$2^{315360*10^8}$	$2^{31556736*10^8}$
$n^{1/2}$	10^{12}	$36*10^{14}$	$1296*10^{16}$	$746496*10^{16}$	$6718464*10^{18}$	$994519296*10^{18}$	$995827586973696*10^{16}$
n	10^6	$6*10^7$	$36*10^8$	$864*10^8$	$2592*10^9$	$31536*10^9$	$31556736*10^8$
$n \lg n$	62746	2801417	133378058	2755147513	71870856404	797633893349	68654697441062
n^2	1000	7745	60000	293938	1609968	5615692	56175382
n^3	100	391	1532	4420	13736	31593	146677
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

(2) Indicate for each pair of expressions (A,B) in the table below, whether A is O, o, Ω, ω, or Θ of B. Assume that k≥1, ϵ>0, and c>1 are constants. Your answer should be in the form of the table with "yes" or "no" written in each box.

	A	B	O	o	Ω	ω	Θ
a.	$lg^k n$	n^ϵ	yes	yes	no	no	no
b.	n^k	c^n	yes	yes	no	no	no
c.	$\sqrt{n}$	$n^{sin\ n}$	no	no	no	no	no
d.	2^n	$2^{n/2}$	no	no	yes	yes	no
e.	n^{lgc}	c^{lgn}	yes	no	yes	no	yes
f.	$lg(n!)$	$lg(n^n)$	yes	no	yes	no	yes

(a) Apply L'Hospital's rule repeatedly to see that $\lim_{n\to\infty} \frac{(lgn)^k}{n^\epsilon} = 0$ to conclude that $(lgn)^k = o(n^\epsilon)$.

$$\begin{aligned}
\lim_{n\to\infty} \frac{(lgn)^k}{n^\epsilon} &= \lim_{n\to\infty} \frac{k(lgn)^{k-1}\frac{1}{n}}{\epsilon n^{\epsilon-1}} \\
&= \lim_{n\to\infty} \frac{k(lgn)^{k-1}}{\epsilon n^\epsilon} \\
&= \lim_{n\to\infty} \frac{k\frac{d(lgn)^{k-1}}{dn}}{\epsilon\frac{dn^\epsilon}{dn}} \\
&= \lim_{n\to\infty} \frac{k(k-1)(lgn)^{k-2}\frac{1}{n}}{\epsilon^2 n^{\epsilon-1}}
\end{aligned}$$

After k applications of the rule, we get

$$\lim_{n\to\infty} \frac{k(k-1)(k-2)....1}{\epsilon^k n^\epsilon} = 0$$

(b) Apply L'Hospital's rule repeatedly to see that $\lim_{n\to\infty} \frac{n^k}{c^n} = 0$ to conclude that $n^k = o(c^n)$.

(c) You can visually inspect the plots to see that $n^{sin\ n}$ is an oscillating function. $sin\ n$ oscillates between 1 and -1. When at its maximum value, $n^{sinn} > c\sqrt{n}$ and thus $n^{sin\ n} \neq O(\sqrt{n})$. When $sin\ n$ is at its minimum, $n^{sin\ n} < c\sqrt{n}$ and thus $n^{sin\ n} \neq \Omega(\sqrt{n})$.

(d) $\lim_{n\to\infty} \frac{2^n}{2^{n/2}} = \infty$ and therefore $2^n = \omega(2^{n/2})$.

(e) Recall that $n^{lgc} = c^{lgc}$.

(f) Note $lg(n^n) = nlg(n)$, and using Stirling's formula it is shown in the text that $lg(n!) = \Theta(nlg(n))$.

(3) Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in 8 n^2 steps, while merge sort runs in 64 n log(n) steps.
For which values of n does insertion sort beat merge sort?

Check the numerical solution of n== 8 Log[n] (log is the natural log, ln)

(4) What is the smallest value of n such that an algorithm whose running time is 100 n^2 runs faster than an algorithm whose running time is 2^n on the same machine?

Check the numerical solution of n== 2 Log[n]+Log[100]

(5) show that for any real constants a, b, with b>0, (n+a)^b = Θ(n^b)
(hint: to show that f(n) =Θ(g(n)) you need to show that it is both O(g(n)) and Ω(g(n)).

Solution

By the definition of $\Theta(\cdot)$, we need find the constants c_1, c_2, n_0 such that $0 \le c_1 n^b \le (n+a)^b \le c_2 n^b$ for all $n \ge n_0$.

Note that for large values of n, $n \ge |a|$ we have

$$n + a \le n + |a| \le 2n$$

and for further large values of n, $n \ge 2|a|$, (i.e., $|a| \le \frac{1}{2}n$)

$$n + a \ge n - |a| \ge \frac{1}{2}n$$

.

Thus, when $n \ge 2|a|$, we have

$$0 \le \frac{1}{2}n \le n + a \le 2n$$

.

Since b is a positive constant, we can raise the quantities to the b^{th} power with out affecting the inequality. thus

$$0 \le (\frac{1}{2}n)^b \le (n+a)^b \le (2n)^b$$

$$0 \le (\frac{1}{2})^b n^b \le (n+a)^b \le (2)^b (n)^b$$

Thus, with $c_1 = (\frac{1}{2})^b, c_2 = 2^b$, and $n_0 = 2|a|$ we satisfy the definition.

Simpler solution:

just prove that

$$\lim_{n \to \infty} \frac{(n+a)^b}{n^b} = \text{constant}$$

Text

(6) show that the function lg(n) is polynomially bounded. Show the same for lg(lg(n))

$$\lim_{n\to\infty} \frac{\lg n}{n^c} = \lim_{n\to\infty} \frac{1/n}{cn^{c-1}} = \lim_{n\to\infty} \frac{1}{cn^c} = 0 \quad \forall c > 0$$

$$\lim_{n\to\infty} \frac{\lg\lg n}{n^c} = \lim_{n\to\infty} \frac{\frac{1}{\lg n} 1/n}{cn^{c-1}} = \lim_{n\to\infty} \frac{1}{cn^c \lg n} = 0 \quad \forall c > 0$$

(7) Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?

a) $$0 \le 2^{n+1} \le c2^n \quad c > 0, \quad n \ge n_0$$

YES, for any c> 2

(b) NO, the limit of the ratio diverges

(8) [redacted]

(1[redacted] Express the function $n^3/1000 - 100n^2 - 100n + 3$ in terms of Θ-notation.

$$f(n) = n^3/1000 - 100n^2 - 100n + 3 = \Theta(n^3)$$

Both upper and lower bounded by n^3 for n large enough

(9) Let $$p(n) = \sum_{i=0}^{d} a_i n^i$$

where $a_d > 0$, be a degree-d polynomial in n, and let k be a constant. Use the definitions of the asymptotic notations to prove the following properties.

a. If $k \geq d$, then $p(n) = O(n^k)$.

b. If $k \leq d$, then $p(n) = \Omega(n^k)$.

c. If $k = d$, then $p(n) = \Theta(n^k)$.

d. If $k > d$, then $p(n) = o(n^k)$.

e. If $k < d$, then $p(n) = \omega(n^k)$.

Solution: keep just the largest power (d) and discuss limits.

(10) Rank the following functions by order of growth; that is, find an arrangement g_1, g_2, … of the functions satisfying $g_1 = \Omega(g_2)$, $g_2 = \Omega(g_3)$, ...,

$$n^2, \quad (\sqrt{2})^{\lg n}, \quad n!, \quad (\lg n)^2, \quad \ln n, \quad 2^n, \quad n \lg n, \quad n^3, \quad \ln \ln n$$

Sqrt[2]^lg(n)=(2^(1/2))^lg(n)=2 ^(1/2 lg(n))=2^lg(n^(1/2))=n^(1/2)=Sqrt[n]

Solution:

$$n! > 2^n > n^3 > n^2 > n \lg n > (\sqrt{2})^{\lg n} = \sqrt{n} > \ln n > \ln \ln n$$

n^n

(a^m)^n=a^m n

a lg b=lg (b^a)

Reminder: Stirling approximation

$$n! \sim \sqrt{2\pi n}(\frac{n}{e})^n \sim e^{n \ln n - n}$$

The substitution method for solving recurrences entails two steps:

1. Guess the form of the solution.
2. Use mathematical induction to find the constants and show that the solution works.

(11) Not for the exam, you may skip

(12) Study by substitution

$T(n)=T(n-a)+T(a)+n$

where a is a constant $a>=1$. For simplicity set $a=1$.

Hint: Let's solve the case $a = 1$; I'll leave the generalization to you:

$$\begin{aligned}
T(n) &= n + T(n-1) \\
&= n + (n-1) + T(n-2) \\
&= n + (n-1) + (n-2) + T(n-3) \\
&= \cdots \\
&= n + (n-1) + (n-2) + \cdots + (1) + T(0) \\
&= \frac{n(n+1)}{2} + T(0).
\end{aligned}$$

In a **recursion tree**, each node represents the cost of a single subproblem somewhere in the set of recursive function invocations. We sum the costs within each level of the tree to obtain a set of per-level costs, and then we sum all the per-level costs to determine the total cost of all levels of the recursion. Recursion trees are particularly useful when the recurrence describes the running time of a divide-and-conquer algorithm.

(13-14) Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 3T(\lfloor n/2 \rfloor) + n$.

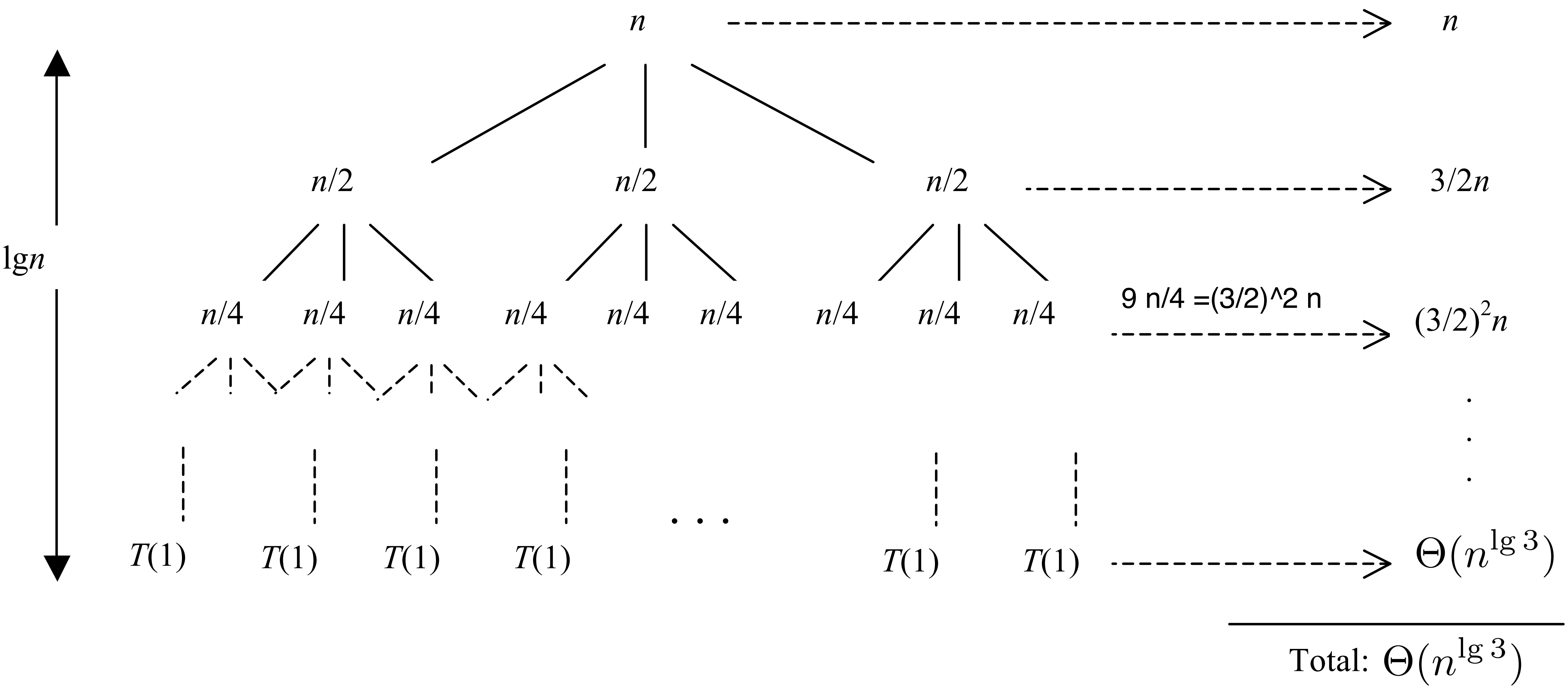
Use the substitution method to verify your answer.

The recurrence is $T(n) = 3T(\lfloor n/2 \rfloor) + n$. We use a recurrence tree to determine the asymptotic upper bound on this recurrence.

Because we know that floors and ceilings usually do not matter when solving recurrences, we create a recurrence tree for the recurrence $T(n) = 3T(n/2) + n$. For convenience, we assume that n is an exact power of 2 so that all subproblem sizes are integers.

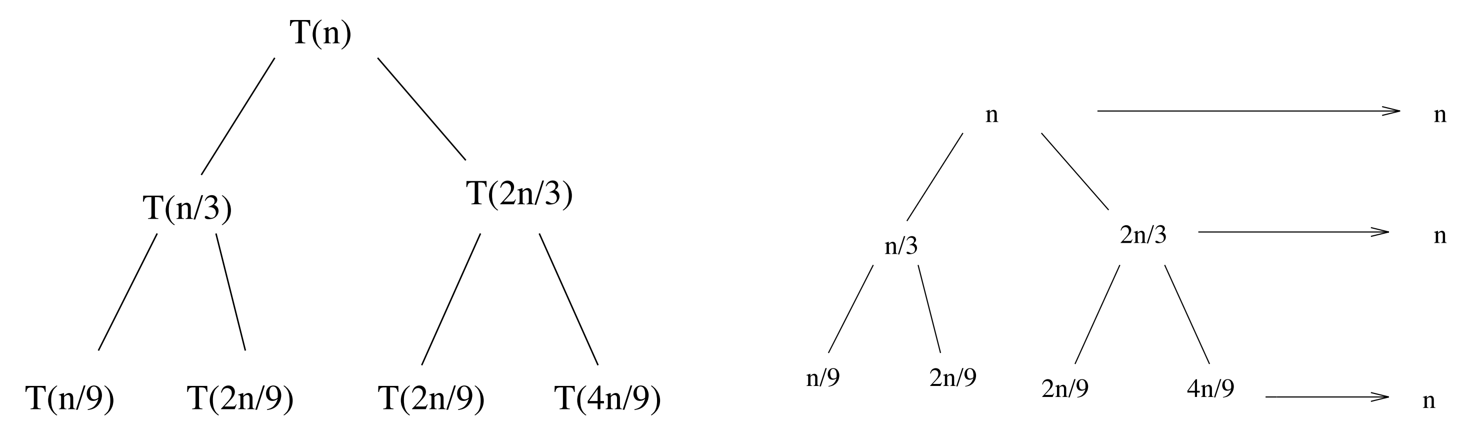
Because subproblem sizes decrease by a factor of 2 each time when go down one level, we eventually must reach a boundary condition $T(1)$. To determine the depth of the tree, we find that the subproblem size for a node at depth i is $n/2^i$. Thus, the subproblem size hits $n = 1$ when $n/2^i = 1$ or, equivalently, when $i = \lg n$. Thus, the tree has $\lg n + 1$ levels (at depth $0, 1, 2, 3, \ldots, \lg n$).

T(n)=3 T(n/2)+n   T(n)=3 (3 T(n/4)+n/2)+n = 9 T(n/4)+3 n/2 +n



(14) Show that the solution to $T(n)=T(n/3)+T(2n/3)+n$ is $\Omega(n \lg n)$ by analyzing the recursion tree.

Draw the recursion tree.



How many levels does the tree have? This is equal to the longest path from the root to a leaf.

The shortest path to a leaf occurs when we take the heavy branch each time. The height k is given by $n(1/3)^k \leq 1$, meaning $n \leq 3^k$ or $k \geq \lg_3 n$.

The longest path to a leaf occurs when we take the light branch each time. The height k is given by $n(2/3)^k \leq 1$, meaning $n \leq (3/2)^k$ or $k \geq \lg_{3/2} n$.

The problem asks to show that $T(n) = \Omega(n \lg n)$, meaning we are looking for a lower bound

On any *full* level, the additive terms sums to n. There are $\log_3 n$ full levels. Thus $T(n) \geq n \log_3 n = \Omega(n \lg n)$

T(n)=4 T(n/2)+3 n lgn

a=4, b=2,   log_b(a)=lg(4)=lg(2^2)=2 lg(2)=2

f(n)=3 n lgn. = O(n^2-eps)

lim_n->inf   n lgn/(n^(2-eps))=0

(15)

The recurrence $T(n) = 7T(n/2)+n^2$ describes the running time of an algorithm A. A competing algorithm A' has a running time of $T'(n) = aT'(n/4) + n^2$. What is the largest integer value for a such that A' is asymptotically faster than A?

Algorithm A: $a = 7\ , \quad b = 2\ , \quad \lg 7 \simeq 2.8$ log_b(a)

2.8- eps >2. 0< eps < 0.8

$$f(n) = n^2 = O(n^{2.8-\epsilon}) \Rightarrow \text{Case 1.} \quad T(n) = \Theta(n^{\lg 7})$$

T(n)= a T(n/4)+n^2

Algorithm A': $a\ , \quad b = 4\ , \quad \log_4 a > 2$ log_b(a) f(n)=n^2 ?= O(n^(Log_b(a)-eps))

>2 Log_4(a) - eps >2. 0< eps <Log_4(a)

$$f(n) = n^2 = O(n^{\log_4 a-\epsilon}) \Rightarrow \text{Case 1.} \quad T(n) = \Theta(n^{\log_4 a})$$

Largest value of a: $\log_4 a = \lg 7 \Rightarrow a \simeq 2^{2\lg 7}$

$$\log_4 a = \frac{\lg a}{\lg 4} = \frac{\lg a}{2}$$

T(n)=Theta(n^Lg(7)) T(n)=Theta(n^Log_4(a)) A' < A, Log_4(a)= lg(7) Log_4(a)= lg(7) a= 4^lg(7)=2^2 lg(7)

a<= 4^lg(7)=2^2 lg(7)=49

(16) Can the master method be applied to the recurrence $T(n) = 4T(n/2) + n^2 \lg n$? Why or why not? Give an asymptotic upper bound for this recurrence.

lg4=2 then yes! Case 2 with a=4,b=2 and k=1

$$\longrightarrow \qquad T(n) = \Theta(n^2(\lg n)^2)$$