(1) For each function **f**(n) and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm takes **f**(n) **microseconds** to solve the problem

Item	1 second	1 miniute	1 hour	1 day	1 month	1 year	1 century
$\lg n$							
$n^{1/2}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

(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}					
b.	n^k	c^n		yes			
c.	$\sqrt{n}$	$n^{\sin n}$					
d.	2^n	$2^{n/2}$					
e.	$n^{\lg c}$	$c^{\lg n}$					
f.	$\lg(n!)$	$\lg(n^n)$					

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

(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?

(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?

(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)).

(7) Prove that the running time of an algorithm is Θ(g(n)) if and only if its worst-case running time is O(g(n)) and its best-case running time is Ω(g(n)). (previous theorem)

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

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

(10) Express the function $n^3/1000 - 100n^2 - 100n + 3$ in terms of Θ-notation.

(11) 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 $p(n) = \omega(n^k)$.

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

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

Stirling approximation for large *n*:

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

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.

(14) Show that the solution to $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ is $O(n \lg n)$.

(15) Study by substitution

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

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

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.

(16) Use a recursion tree to determine a good asymptotic upper bound on the recurrence

$T(n) = 3T(\lfloor n/2 \rfloor) + n.$

Verify your answer by the substitution method.

(17) Show that the solution to

$T(n)=T(n/3)+T(2n/3)+n$

is $\Omega(n \lg n)$ by analyzing the recursion tree.

The master method provides a "cookbook" method for solving recurrences of the form

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function. The master method requires memorization of three cases, but then the solution of many recurrences can be determined quite easily, often without pencil and paper.

The recurrence

$$T(n) = aT(n/b) + f(n) .$$

describes the running time of an algorithm that divides a problem of size n into a subproblems, each of size n/b, where a and b are positive constants. The a subproblems are solved recursively, each in time T (n/b). The cost of dividing the problem and combining the results of the subproblems is described by the function f (n).

As a matter of technical correctness, the recurrence isn't actually well defined because n/b might not be an integer. Replacing each of the a terms T (n/b) with either $T(\lfloor n/b \rfloor)$ or $T(\lceil n/b \rceil)$ doesn't affect the asymptotic behavior of the recurrence, however.

(18) 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?

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