1. Big-O, Little-o, Little-omega, Big-Theta (No Justification Necessary)

For each example, indicate whether $f = o(g)$ (little-oh), $f = \omega(g)$ (little-omega), or $f = \Theta(g)$ (big-Theta). No justification is necessary.

(a) $f(n) = 2n^2 + 7n$, $g(n) = n^3$
(b) $f(n) = n \log n$, $g(n) = n^2$
(c) $f(n) = 10^n$, $g(n) = 2^n$
(d) $f(n) = \sum_{i=1}^{n} i^2$, $g(n) = n^3$
(e) $f(n) = \log_2 n$, $g(n) = \log_{10} n$

2. Formal Proof of Logarithmic Big-O

Give a formal proof of the following statement: If $f(n) \geq 1$ for all $n \in N$, $g(n) \geq 1$ for all $n \in N$, $f(n) = O(g(n))$, and $g(n)$ is unbounded (meaning $\lim_{n\to\infty} g(n) = \infty$) then $\log_2 f(n) = O(\log g(n))$.

Use the formal definition of big-Oh in your answer. In your proof, you can use the fact that the value of $\log_2 n$ increases as n increases. You may also assume that $f(n)$ is unbounded.

Note that a similar statement with exponential functions is not true: If $f(n) = 2^n$ and $g(n) = n^2$, then $f(n) = O(g(n))$ is false, but 2^{2^n} is not $O(2^{n^2})$.

3. Applying Our Master Theorem

For each of the following recurrences, determine whether Our Master Theorem (on the last page of this HW) can be applied to the recurrence. If it can, use it to give the solution to the recurrence in Θ notation; no need to give any details. If not, write "Our Master Theorem does not apply."

(a) $T(n) = 2T(n/2) + n \log n$
(b) $T(n) = 9T(n/3) + n^2$
(c) $T(n) = T(n/2) + 1$

4. Analysis of Modified Recurrence

Our Master Theorem can be applied to a recurrence of the form $T(n) = aT(n/b) + n^d$, where a, b, d are constants with $a > 0$, $b > 1$, $d > 0$. Consider instead a recurrence of the form $T_{new}(n) = aT_{new}(n/b) + n^d \log n$ where $a > 0$, $b > 1$, $d > 0$ (and $T(1) = 1$).

For each of the following, state whether the given property of T_{new} is true. If so, explain why it is true. If not, explain why it is not true. (Even if you know the version of the Master Theorem in the textbook, don't use it in your explanation.)

(a) $T_{new}(n) = O(n^{d+1})$ if $\log_b a < d + 1$
(b) $T_{new}(n) = \Omega(n^{\log_b a} \log n)$

5. Recurrence Relation Analysis

Consider the recurrence $T(n) = 2T(n/2) + n \log n$ for $n > 1$, and $T(1) = 1$.

(a) Compute the value of $T(4)$, using the recurrence. Show your work.
(b) Use a recursion tree to solve the recurrence and get a closed-form expression for $T(n)$, when n is a power of 2. (Check that your expression is correct by plugging in $n = 4$ and comparing with your answer to (a).)
(c) Suppose that the base case is $T(2) = 3$, instead of $T(1) = 1$. What is the solution to the recurrence in this case, for $n \geq 2$?

6. Modified Mergesort

Consider a variation of mergesort that works as follows: If the array has size 1, return. Otherwise, divide the array into fourths, rather than in half. Recursively sort each fourth using this variation of mergesort. Then merge the first (leftmost) fourth with the second fourth. Next, merge the third fourth with the fourth fourth. Finally, merge the first two merged portions.

(a) Write a recurrence for the running time of this variation of mergesort. It should be similar to the recurrence for ordinary mergesort. Assume n is a power of 4.
(b) Apply Our Master Theorem to the recurrence to get the running time of the algorithm, in theta notation. Show your work.

7. Recursive Sorting Algorithm

Consider the following recursive sorting algorithm. Assume n is a power of 2. (Note: This is not a version of mergesort. No merges are performed.)

- If the array has only one element, return.
- Recursively sort the first half of the elements in the array.
- Recursively sort the second half of the elements in the array.
- Recursively sort the first half of the elements in the array again.
- Recursively sort the second half of the elements in the array again.

(Note: Even if you can't figure out part (a) below, you can still answer (b) and (c).)

(a) Prove that the algorithm is correct by showing that the array will be sorted after the four recursive calls are performed, assuming the four recursive calls correctly sort their (sub)arrays. Note that each half of the array is included in two of the 4 recursive calls.
(b) Write a recurrence expressing the running time of the algorithm.
(c) Apply Our Master Theorem to your recurrence. What is the running time of the algorithm, in theta notation?

Our Master Theorem

Let a, b, d, n_0 be constants such that $a > 0$, $b > 1$, $d \geq 0$ and $n_0 > 0$. Let $T(n) = aT(n/b) + \Theta(n^d)$ for when $n \geq n_0$, and $T(n) = \Theta(1)$ when $0 \leq n < n_0$. Then,

$$T(n) = \begin{cases} \Theta(n^d \log n) & \text{if } d = \log_b a \\ \Theta(n^{\log_b a}) & \text{if } d < \log_b a \\ \Theta(n^d) & \text{if } d > \log_b a \end{cases}$$

We assume here that $T(n)$ is a function defined on the natural numbers. We use $aT(n/b)$ to mean $a'T(\lfloor n/b \rfloor) + a''T(\lceil n/b \rceil)$ where $a', a'' > 0$ such that $a' + a'' = a$.