Q1) *The asymptotic runtime costs are written down as expressions the form O(n), Theta(n^2 lg(n)), Omega(2^n) etc. What value for n should one use when comparing such expressions?*

A) Let us say that f(n) and g(n) are functions of n (such as n, n^2, n^3, lg(n), n^2 lg(n), etc). When comparing, O(f(n)) with O(g(n)), you should consider a “sufficiently large” value of n. This means a very large (integer) value of n, approaching infinity. In other words, if f(n) gives a value less than g(n) for small n’s, but after a large enough number f(n) is always greater than g(n), we can ignore the smaller values of n, and say that O(f(n)) is greater than O(g(n)).

The same holds when you are comparing Theta- and Omega- asymptotic expressions.

As a side note, the three asymptotic notations (O, Theta, Omega) express different information about the running time of an algorithm. You should read more about this from the notes, but in a nutshell, big-Oh expresses an upper bound of the running time; Omega expresses a lower bound; and Theta expresses a precise bound (both upper and lower bound).

Q2) *Slide 20 of Lecture 4 contains a number of arithmetic operations between expressions using the Theta notation. Do the same operations hold when we use the big-Oh, or even the Omega, notation?*

A) Yes. The same rules for simplification, addition, and multiplication hold for big-Oh expressions. Also the same hold for Omega expressions.

Q3) *There seems to be a theme where the 2nd part of Question 1(a) asks to describe an improved implementation of an algorithm that provides a certain amortized running time. Assuming this trend continues, what does describe mean? Are you looking for a word description or actual code or both?*

A) When a question asks you to “describe” or “explain" an algorithm or implementation strategy, it is only asking for a relatively short English text doing exactly that: describe the algorithm. You are of course free you use code snippets to explain something, but usually this does not make the explanation any easier. You only need to write code in questions that explicitly ask you to “implement” something.

Q4) *When using hash tables, dealing with collisions gives worst case running times of O(n) or O(n/k), depending on whether we use linear probing or separate chaining. As k is a constant, this really gives an O(n) worst-case running time for all operations of a hash table, no matter how we implement it.*

a) *If these are the run times, what is the value of using a hash table as opposed to a linked list?*

A) Indeed, hash tables have terrible worst-case running times — they are as bad (or as good) as linked lists.
However, hash tables have an excellent *average* running time! If we assume that the keys we insert in them have “evenly distributed hashes” (uniform hashing assumption), then the average running time of every insert/lookup operation is O(1)!

b) *I have also been led to believe that in theory a hash table could have a run time of O(1), how could one prove this?*

A) It is not a matter of theory and practice. It is a matter of worst-case and average-case running times of this data structure, as mentioned above.

Q5) *When a question asks for writing the "main operations in a specific API and a description", should we write just the title of the methods with a description?*

A) You should write:

- The name of the class, together with any type parameters, if the class is generic
- The signature of the main constructor(s) and methods, with a one-line description. A signature of a method/constructor contains its name, its arguments together with their types, and the return type. The description should state what the method does at a high level. There are many examples of this in the notes and book.

Q6) One more last minute question:

Sorry to email so late but could you clarify the difference between big Oh, big theta and big omega with regards to questions such as 2016 Question 2 (a) and 2015 Question 2 (a).

My understanding is that as big theta, big omega and big O are all forms of asymptotic bounds they can all be simplified through elimination of multiplicative constants.

This is correct

Should we treat run-times of Big O, Big theta and Big Omega any differently beyond this when ordering them?

For Example;
$\Theta(n \log n) > \Theta(n)$
$\Theta(n^2 + 3n - 1) = \Theta(n2)$
$\Theta(1) = \Theta(10)$
$\Theta(5n) < \Theta(n^2)$
$\Theta(n^3 + \log(n)) = \Theta(100n^3 + \log(n))$
$\Theta(1) = \Theta(10) < \Theta(5n) = \Theta(n) < \Theta(n \log n) < \Theta(n^2) = \Theta(n^2 + 3n - 1) < \Theta(n^3 + \log(n)) = \Theta(100.n^3 + \log(n))$
Is this solution correct?

This is correct.

and would the solution be the same if run times were expressed in Big O notation?

Yes.

My main source of confusion came from a statement made on stack overflow: "Basically when we say an algorithm is of O(n), it's also O(n2), O(n1000000), O(2n), ... but a Θ(n) algorithm is not Θ(n2)." Full thread here

It is difficult to try to explain partial explanations from the web, but I'll attempt it here because it might help many people understand asymptotic notation better:

As the student wrote above, Θ(), O() and Ω() express asymptotic *bounds*.

Let us assume an algorithm with a running time T(n), which is a function of the algorithm's input size n.

When we say that T(n) is O(n), it roughly[[*]] means that T(n) is less than or equal to the function f(n)=n. In other words, when we say that T(n) is O(n), we give an *upper bound* for T(n). For better or for worse, we write this upper-bound statement by abusing the mathematical equals symbol: T(n)=O(n).

[[I say "roughly" because we are allowed to consider only very large values of n, and to ignore all multiplicative constants and all lower-order factors]]

Note that this equals symbol does not mean "equals" as we understand it from simple math. It is a shorthand for saying that "T(n) is a function that belongs in the complexity class O(n)". But we did not dive into the gory math details of complexity classes and the definitions of O, Theta and Omega, and I don't think you need to understand this math to use it in practice. In practice, the only thing we need to understand when we write T(n)=O(n), or when we say "the worst case running time of this algorithm is O(n)" is that, when the input size is K, the algorithm will run in time less than or equal to K [[up to all the simplifications mentioned above]].

And here is the confusing part: when O(n) is an upper bound of T(n), it also follows that O(n^2) is an upper bound of T(n). So, confusingly, if T(n)=O(n), we can also write T(n)=O(n^2), again by abusing the equals symbol.

Similarly, if T(n)=O(n) we can also write that T(n)=O(n^3), etc.

Now back to the questions in the past exams: these questions were asking to compare the *asymptotic bounds* themselves. Obviously, the O(n) upper bound is a smaller upper bound than the O(n^2) upper bound. Therefore we can write O(n) < O(n^3). Moreover, the O(n) upper bound is the same as the O(10.n + 5) upper bound [[because of the simplifications above]], and we can write O(n) = O(10.n + 5). Here the equals symbol is the standard one -- O(n) is mathematically the same as O(10.n + 5).

The same story as above holds for the Omega (Ω) asymptotic notation, with the only difference that with the Omega notation we express *lower bounds*. For example, when T(n)=Ω(n), we mean that T(n) is greater than or equals to f(n)=n [[up to the simplification above]], which also follows that T(n) is greater than or equals to g(n)=log n), which means that T(n)=Ω(log n). However, Ω(log n) < Ω(n).

It is also the same with Theta (Θ), but because the Theta notation expresses precise bounds, if T(n)=Θ(n), then
T(n) is *not* equal to Θ(log n) and
T(n) is *not* equal to Θ(n^2).

Here precise bound means that when we write T(n)=Θ(n) it are really writing "T(n)=O(n) and T(n)=Ω(n)".