1

a) It would try to divide by 0 because part of the code is 1.0/delta.
b) As the 16-bit delta gets smaller, the sum gets smaller. Starting at 1 for 10^-1 and continuing to .03 at 10^-5. For the 32-bit, the behavior is a little bit more curious. It starts at 1 like the 16-bit, then goes down to .9999 by the 10^-3. Then, in 10^-4 and 10^-5, it increases above 1. The 64-bit behavior is a little more straightforward. The sum stays at 1.0 until 10^-5, when it becomes .999999998.

c) c) 16-bit error occurs at 10^-8. 32-bit: 10^-46. 64-bit: 10^-309

2 The roots make sense for the first two functions. However, problems arise in the last two. First, in the second to last, there are so many roots, it's not effective to try and find every one – especially if your brackets include 0, where the majority of the roots appear to close-in on. Second, the last one does not work because it's dividing by zero because the asymptote is on y=0 therefore there are no roots. I have avoided evaluating the last one because it produces an error.

Maximum number of iterations should be based off the equation n = log2(|a-b|/E) which uses tolerance. Thus, the number of iterations you use shouldn't have to be any more than n.

4. I don't know if I should write this, because you will probably take off points, but does our tolerance really have to be so small for this one. I mean, it's not microscopic, so even one or two decimal points should be okay, right? Like I said, take off all the points you want. The long hope I might receive a response is worth it.