MysteryFunction1: calculates the value of “a” raised to the power of “b”

MysteryFunction2: This function examines the 32 bits of the parameter “num” from the least significant bit to the most significant bit. For each bit that is set to 1, it sets the corresponding bit, from the value -2147483648, in the result r. A new number is created where only the most significant bit of each set position is set. The zeros from num are ignored and the corresponding generated number will set the corresponding bit by significance i.e. when looking at the 2^1 if the bit is 1, then the result will get store a zero in the result because the first bit in the 2^1 was 0. This will process will continue the process up to 32 times. 1 iteration for each bit in the resulting int. I think this might be a pseudo random number generator.

MysteryFunction3: This function is looking for the max in the array.

MysteryFunction4: The function examines the bits in the binary value of the number provided in the parameter “n”. It keeps a count of all of the bits that are set to 1 e.g. if n is 1010 base 2 (10 decimal) then the returned value will be 2 because there are 2 bits set to 1 in the number.

MysteryFunction5: finds differing bits between “A” and “B” and tallies number of bits.

2. Discussion on the approach(es) that you used to identify what the mystery function does.

My strategy for each of the mystery functions to get the C code was to take the assembly version and get a working example using “as” and “gcc”. Once I had the exact way of testing c code for correctness, I wrote a C function that was a line-by-line representation of the assembly code as best as possible using a combination of “goto” and “if” statements for jumps. Then I looked at the moving pieces and translated it to more c like code e.g. goto/if combos were change to while or if/else blocks respectively. After that it was a matter of testing the functions to look for patterns in the output and in the code to identify the work the code is doing.