

$$\begin{aligned}
 1. \quad f(x) &= (x^{2.3} + x \ln x^5) (1.1^{x+1} + 1.2^x) + (x^2 + 1.2^x) (x^3 + .92^x) \\
 &\rightarrow O(x^{2.3} + 5x \ln x) = O(x^{2.3}) \\
 &\rightarrow O(1.1^{x+1} + 1.2^x) = O(1.2^x) \\
 &\rightarrow O(x^2 + 1.2^x) = O(1.2^x) \\
 &\rightarrow O(x^3 + .92^x) = O(x^3)
 \end{aligned}$$

$$\begin{aligned}
 &O(x^{2.3}) * O(1.2^x) + O(1.2^x) * O(.92^x) \\
 &O(x^{2.3} * 1.2^x) = g(x)
 \end{aligned}$$

2. No.

Consider $x = 6$:

$$\frac{x^2 6^x}{6^x} = x^2, \text{ which cannot be upper bound by any constant } k. \text{ Therefore } x^2 6^x \text{ is not } O(6^x)$$

Now consider $x > 6$:

$$\lim_{x \rightarrow \infty} \left(\frac{x^2 6^x}{a^x} \right) = 0, \text{ which shows there is an upper bound by some constant } k. \text{ Therefore } x^2 6^x \text{ is in } O(6^x).$$

However, There are infinite real numbers such that $a > 6$, therefore there is no smallest real number that places $x^2 6^x$ in $O(6^x)$.

$$3. \quad CA10_{16} + 4F57_{16}:$$

$$7 + 0 = 7$$

$$5 + 1 = 6$$

$$A + F = 19 \quad \text{Carry the 1 to the next line}$$

$$1 + C + 4 = 11$$

$$CA10_{16} + 4F57_{16} = \mathbf{11967}_{16}$$

$$4. \quad 11011_2 * 1001_2: (x \text{ is placeholder})$$

$$\begin{array}{r} \text{xxx}11011 \\ \text{xx}000000 \\ \text{x}0000000 \\ \hline 11011000 \\ \hline 11110011 \end{array}$$

$$\mathbf{11110011} \quad \text{ANSWER}$$

$$\begin{aligned}
5. \quad & 621 = 82 * 7 + 47 \\
& 82 = 47 * 1 + 35 \\
& 47 = 35 * 1 + 12 \\
& 35 = 12 * 2 + 11 \\
& 12 = 11 * 1 + 1 \\
& 11 = 1 * 11 + 0
\end{aligned}$$

The last nonzero remainder is 1, so by the Euclidean Algorithm, the integers **621 and 82 are relatively prime.**

$$\begin{aligned}
6. \quad n * 35 &= 32n + 3n \\
&= 2^5n + 3n \\
&= 2^5n + 2n + n
\end{aligned}$$

$$\text{Mem} = 2^5n + 2n$$

$$\text{Result} = \text{Mem} + n$$

7. Just like multiplying a decimal number by 10_{10} , when multiplying an octal number by 10_8 (8_{10}) you simply add a zero to the right side of the number like a bit-shift. In the instance of $n = 741_8$,

$$n * 10_8 = 7410_8$$

8. The division in this case is the opposite direction of shift of multiplication, perform the “bit-shift” when dividing with powers of 10_b . The cut bit becomes the remainder. In the instance of $n = 741_8$,

$$\frac{741_8}{10_8} = 74 \text{ with a remainder of } 1.$$

$$\begin{aligned}
9. \quad n = 0 : 3^{2^0} &= 3^1 = 3 \bmod(13) \\
n = 1 : 3^{2^1} &= 3^2 = 9 \bmod(13) \\
n = 2 : 3^{2^2} &= 3^4 = 81 \bmod(13)
\end{aligned}$$

$$\text{If } n \text{ is odd then : } 3^{2^n} = 9 \bmod(13)$$

$$\text{If } n \text{ is even then : } 3^{2^n} = 3 \bmod(13)$$

10. Calculating hex numbers A...A requires a geometric summation. This can be defined by:

In Hex:

$$\sum_0^n 0xA * 0x10^n = 0xA \sum_0^n 0x10^n$$

In Decimal:

$$\sum_0^n 10 * 16^n = 10 \sum_0^n 16^n$$

This simplifies by the Geometric Sum Formula to:

$$10 \frac{16^n - 1}{16 - 1} = 10 \frac{16^n - 1}{15} = 2 \frac{16^n - 1}{3}$$

11. EXTRA CREDIT:

```

1 # header
2 print("Name: William Jedynak")
3 print("ID : 1227139214\n")
4
5 def oneToOne(lst): # check if function is one-to-one
6     for i in range(len(lst)):
7         for j in range(len(lst)):
8             if i != j and lst[i] == lst[j]:
9                 return False
10        return True
11
12    lst = []
13    for a in range(1, 101):
14        lst.clear()
15        for x in range(1, 101):
16            lst.append(a ** x % 101)
17            if oneToOne(lst):
18                print(str(a) + "^" + str(x) + " mod 101 is bijective")
19    for a in range(1, 101):
20

```

Output:

```

"C:\Users\WillJedynak\OneDrive\ASU Online\Fall 2022\Session B\MAT
Name: William Jedynak
ID : 1227139214

2^100 mod 101 is bijective
3^100 mod 101 is bijective
7^100 mod 101 is bijective
8^100 mod 101 is bijective
11^100 mod 101 is bijective
12^100 mod 101 is bijective
15^100 mod 101 is bijective
18^100 mod 101 is bijective
26^100 mod 101 is bijective
27^100 mod 101 is bijective
28^100 mod 101 is bijective
29^100 mod 101 is bijective
34^100 mod 101 is bijective
35^100 mod 101 is bijective
38^100 mod 101 is bijective
40^100 mod 101 is bijective
42^100 mod 101 is bijective
46^100 mod 101 is bijective
48^100 mod 101 is bijective

```

Run: Main

```

42^100 mod 101 is bijective
46^100 mod 101 is bijective
48^100 mod 101 is bijective
50^100 mod 101 is bijective
51^100 mod 101 is bijective
53^100 mod 101 is bijective
55^100 mod 101 is bijective
59^100 mod 101 is bijective
61^100 mod 101 is bijective
63^100 mod 101 is bijective
66^100 mod 101 is bijective
67^100 mod 101 is bijective
72^100 mod 101 is bijective
73^100 mod 101 is bijective
74^100 mod 101 is bijective
75^100 mod 101 is bijective
83^100 mod 101 is bijective
86^100 mod 101 is bijective
89^100 mod 101 is bijective
90^100 mod 101 is bijective
93^100 mod 101 is bijective
94^100 mod 101 is bijective
98^100 mod 101 is bijective
99^100 mod 101 is bijective

Process finished with exit code 0

```