Completed without assistance — Andrew Pan

1. a. 1. No, because there is no x that loves all y
2. Yes, because there is a y (c) that all x's love
3. Yes, because all x love at least one y
4. Yes, because all y have at least one x that loves y
5. Yes, because there is at least one x that loves at least one y
6. No, because all x love at least one y
7. No, because all x love at least one y

b.

Statement	Table 1	$x \geq y$	$x = y$
$\forall x, x,\ loves(x,x)$	False	True	True
$\forall x,y,\ loves(x,y) \Rightarrow loves(y,x)$	False	False	True
$\forall x,y,z,\ loves(x,y) \wedge loves(y,z) \Rightarrow loves(x,z)$	False	True	True

2. 1. ~~$sum(0,0,0) \rightarrow sum(0,$~~
4. $\forall x.\ sum(x,0,x) \rightarrow sum(x, s(0), s(x)), 1, 2$
5. $x = 0 \rightarrow sum(s(0), s(0), s(s(0))), 4$
6. $\exists x,\ sum(x, x, s(s(0))), 5$

2, 4. $\forall x.\ sum(x,0,x), 1$
5. $x = 0 \rightarrow sum(0,0,0) \rightarrow sum(0, s(0), s(0)), 4, 2$
6. $\forall n.\ sum(0, s(0), s(0)) \rightarrow sum(0, s^n(0), s^n(0)), 5$
7. $\forall x \in \{\forall n \in \{s^n(0)\}\}$
8. $\forall x.\ sum(0, x, x), 7, 6$

3. a. (max_ending_here, max_so_far):
$(0,0) \rightarrow (1,1) \rightarrow (0,1) \rightarrow (4,4) \rightarrow (3,4) \rightarrow (5,5) \rightarrow (6,6) \rightarrow (1,6) \rightarrow (5,6)$

b. No because max_subarray iterates through the list instead of breaking the list into smaller parts

c. Yes because max_subarray breaks the problem down into finding the maximum of two numbers multiple times

d. For this implementation of max_subarray(), memoization does not change the operation of the function for the given list, and does not reduce the order of growth.

4. a. bin(3, lst, 0, 4) → bin(3, lst, 0, 4) → bin(3, lst, 0, 2) → ~~bin(3, lst, 0, 2)~~ |

b. Yes because it divides the current sublist it is looking at into 2 sublists with each recursive call

c. No, because a normal search would still use comparisons, as binary_search_array does.

d. For this implementation of binary_search_array memoization does not change the operation of the function for the given list, and does not reduce the order of growth.

a. Memoization helps if the same value is being searched for in the same sublist over multiple runs, but provides no benefit for a single call. It relates to the recursion graph by eliminating further recursive calls for a ~~memoized~~ call that has already been memoized, and to dynamic programming by reducing a comparison operation to looking for the key-value pair in a dictionary instead.