Problem A

Initial values for the array = [ {0, 1, 0}, {1, 0, 1} ]

When rotate() is called, since values[0].z is 0, the net result of the call is rotate(values, values[0]).

Inside rotate(), a **copy** of values array and a **copy** of values[0] (which is {x: 0, y: 1, z: 0}) are passed into the function. I’ll mark the copy of values[0] as b\_copy to indicate it's a local copy inside the function:

1. a[0].x >= a[1].x: This checks if 0 >= 1, which is false, so b\_copy.y becomes a[1].y, which is 0.
2. a[0].y > a[1].y: This checks if 1 > 0, which is true, so b\_copy.z becomes a[0].z, which is 0.
3. a[0].z <= a[1].z: This checks if 0 <= 1, which is true, so b\_copy.x becomes a[0].x, which is 0.

So, after rotate(), b\_copy (copy of values[0] inside the function) is {x: 0, y: 0, z: 0}.

However, since we are operating under call-by-value for Part A, b\_copy is just a local copy, and changes to b\_copy do not affect the original values array outside of the function. Therefore, the values array remains unchanged. So, we’ve done all that work for nothing.

Final values for the array = [ {0, 1, 0}, {1, 0, 1} ]

Problem B

Initial values for the array = [ {0, 1, 0}, {1, 0, 1} ]

The rotate function is called with values and values[values[0].z] as arguments. Given values[0].z is 0, this means b is essentially values[0].

Inside rotate():

1. if (a[0].x >= a[1].x): This checks if 0 >= 1, which is false. So, b.y = a[1].y; makes values[0].y = 0;.
2. if (a[0].y > a[1].y): This checks if 0 > 0, which is false. So, b.z = a[1].z; makes values[0].z = 1;.
3. if (a[0].z <= a[1].z): After the second step, this checks if 1 <= 1, which is true. So, b.x = a[0].x; keeps values[0].x = 0;.

Final values for the array = [ {0, 0, 1}, {1, 0, 1} ]

Problem C

Initial values for the array = [ {0, 1, 0}, {1, 0, 1} ]

The function rotate modifies a copy of values[0] (since values[values[0].z] resolves to values[0] as values[0].z is 0).

Inside rotate():

1. Y-axis decision: Since a[0].x is not greater than or equal to a[1].x (0 >= 1 is false), it goes to the else condition, setting b.y (or the copied values[0].y) to a[1].y, which is 0. This does not alter the y-value as it remains 0.
2. Z-axis decision: Since a[0].y is not greater than a[1].y (0 > 0 is false), it sets b.z (or the copied values[0].z) to a[1].z, which is 1.
3. X-axis decision: Given the update from the previous step, a[0].z becomes 1. Since a[0].z is less than or equal to a[1].z (1 <= 1 is true), it attempts to set b.x (or the copied values[0].x) to a[0].x, which is already 0. No change here.

Final values for the array = [ {0, 0, 1}, {1, 0, 1} ]

Problem D

Initial values for the array = [ {0, 1, 0}, {1, 0, 1} ]

Call-by-name means that wherever a is used in rotate(), it refers to values, and wherever b is used, it refers to values[values[0].z], which initially is values[0].

Inside rotate():

1. b.y assignment:
   1. The condition a[0].x >= a[1].x (i.e., 0 >= 1) is false, so b.y = a[1].y, which results in values[0].y being set to values[1].y (0).
2. b.z assignment:
   1. After the first step, since b is essentially values[0], we re-evaluate values[0] every time b is mentioned. Now, values[0].y has been changed to 0.
   2. The condition a[0].y > a[1].y (i.e., 0 > 0) is false (note this is a change due to the previous step affecting the evaluation), so b.z = a[1].z, setting values[0].z to values[1].z (1).
3. b.x assignment:
   1. Again, considering b as values[0], the condition a[0].z <= a[1].z is now 1 <= 1, which is true. Thus, b.x = a[0].x, but since values[0].x is already 0, there's effectively no change here.

Final values for the array = [ {0, 0, 1}, {1, 0, 1} ]