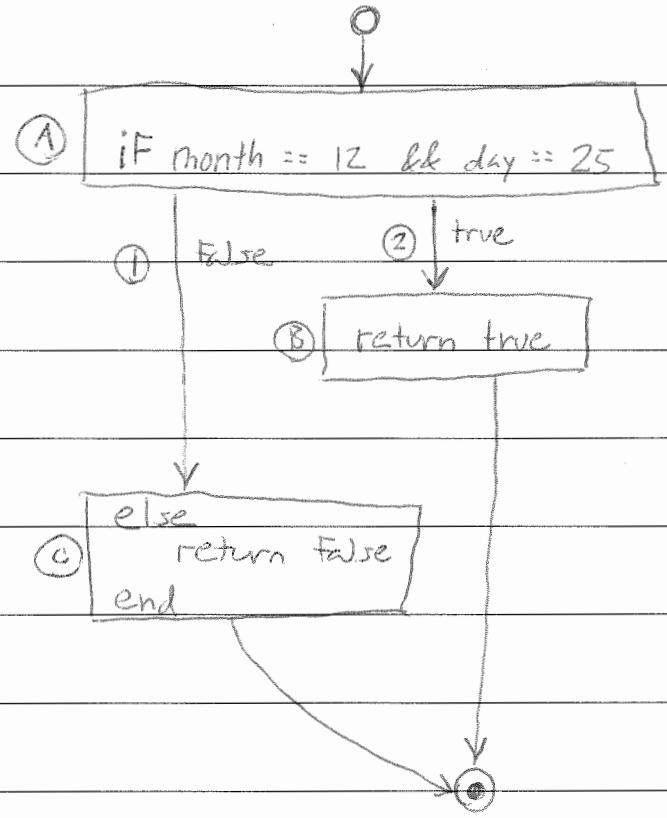
Consider this function.

```
def is_it_xmas?(month, day)
    if month == 12 && day == 25
        return true
    else
        return false
    end
end
```

Draw a control-flow graph for the function. In addition to the usual CFG features, label the nodes with capital letters (A, B, C, etc.), and label the edges with numbers (1, 2, 3, etc.).



Fill in the table below with a test suite that provides statement coverage. In the Covers column, list the letter labels (A, B, C, etc.) of the nodes covered by each test case.

Input		Expected Output	Covers
month	day		
12	24	false	A, C
12	25	true	A, B

The function is correct to the best of my knowledge.

What change to a line in the function would introduce a bug that your above test suite catches?

day == 25 → day == 24

What change to a line in the function would introduce a bug that your above test suite does not catch?

day == 25 → day >= 25

Fill in the table below with a test suite that provides branch coverage. In the Covers column, list the number labels (1, 2, 3, etc.) of the edges covered by each test case (only true/false edges needed).

Input		Expected Output	Covers
month	day		
12	24	false	1
12	25	true	2

The function is correct to the best of my knowledge.

What change to a line in the function would introduce a bug that your above test suite catches?

day == 25 ⟶ day == 24

What change to a line in the function would introduce a bug that your above test suite does not catch?

day == 25 ⟶ day >= 25

Fill in the table below with a test suite that provides path coverage. In the Covers column, list the number labels (1, 2, 3, etc.) of the edges covered by each test case. You need only cover executions that involve at most 1 iteration of each loop (if there are any).

Input		Expected Output	Covers
month	day		
11	24	false	1
11	25	true	2
			Paths: - 1
			- 2

The function is correct to the best of my knowledge.

What change to a line in the function would introduce a bug that your above test suite catches?

day == 25 → day == 24

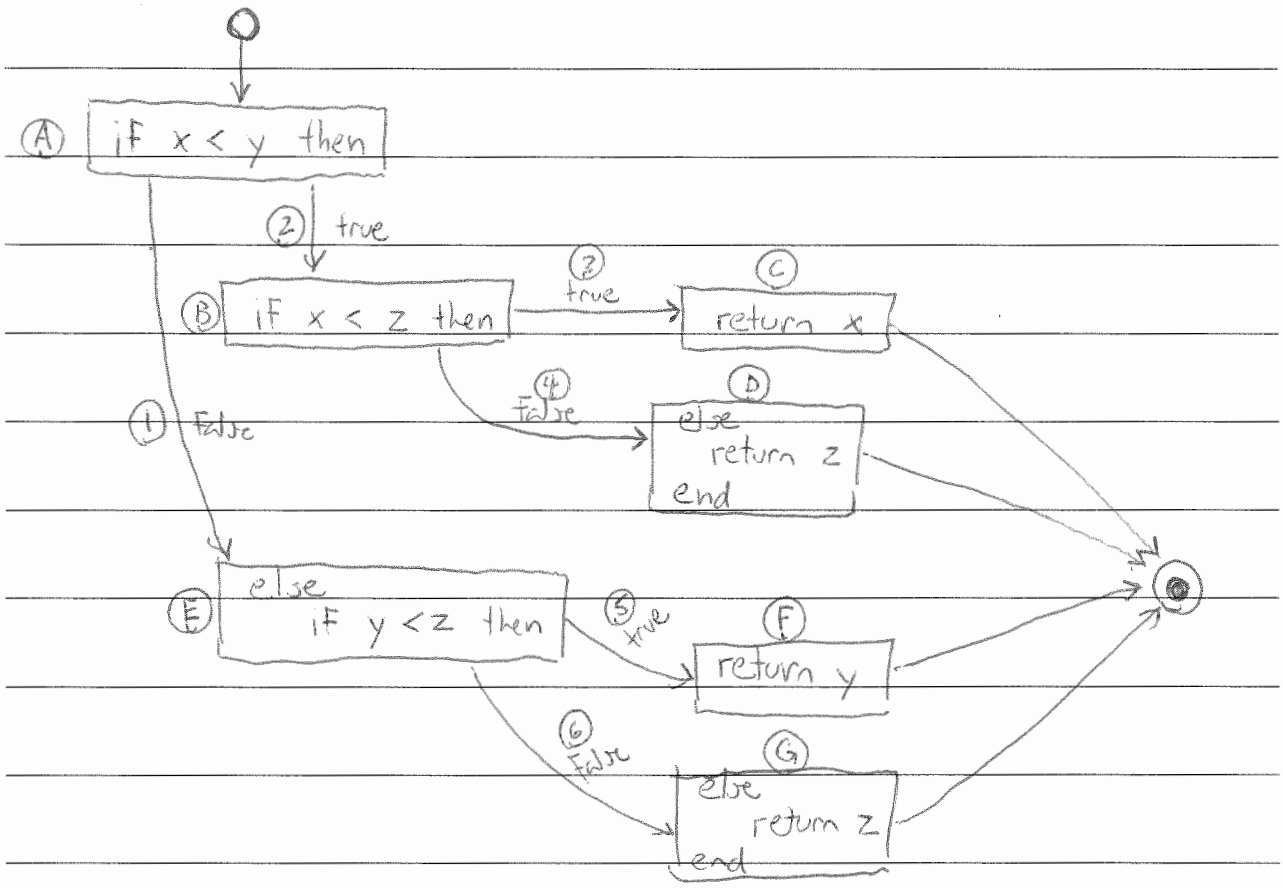
What change to a line in the function would introduce a bug that your above test suite does not catch?

day == 25 → day >= 25

Consider this function.

```
def min_of_three(x, y, z)
    if x < y then
        if x < z then
            return x
        else
            return z
        end
    else
        if y < z then
            return y
        else
            return z
        end
    end
end
```

Draw a control-flow graph for the function. In addition to the usual CFG features, label the nodes with capital letters (A, B, C, etc.), and label the edges with numbers (1, 2, 3, etc.).



Fill in the table below with a test suite that provides statement coverage. In the Covers column, list the letter labels (A, B, C, etc.) of the nodes covered by each test case.

Input			Expected Output	Covers
x	y	z		
1	2	3	1	A, B, C
1	2	0	0	A, B, D
2	1	3	1	A, E, F
2	1	0	0	A, E, G

The function is correct to the best of my knowledge.

What change to a line in the function would introduce a bug that your above test suite catches?

x < y → x > y

What change to a line in the function would introduce a bug that your above test suite does not catch?

return x → return y - x

Fill in the table below with a test suite that provides branch coverage. In the Covers column, list the number labels (1, 2, 3, etc.) of the edges covered by each test case (only true/false edges needed).

Input			Expected Output	Covers
x	y	z		
1	2	3	1	2, 3
1	2	0	0	2, 4
2	1	3	1	1, 5
2	1	0	0	1, 6

The function is correct to the best of my knowledge.

What change to a line in the function would introduce a bug that your above test suite catches?

x < y → x > y

What change to a line in the function would introduce a bug that your above test suite does not catch?

return x → return y - x

Fill in the table below with a test suite that provides path coverage. In the Covers column, list the number labels (1, 2, 3, etc.) of the edges covered by each test case. You need only cover executions that involve at most 1 iteration of each loop (if there are any).

Input			Expected Output	Covers
x	y	z		
1	2	3	1	2,3
1	2	0	0	2,4
2	1	3	1	1,5
2	1	0	0	1,6
				Paths - 2,3 - 2,4
				- 1,5 - 1,6

The function is correct to the best of my knowledge.

What change to a line in the function would introduce a bug that your above test suite catches?

x < y → x > y

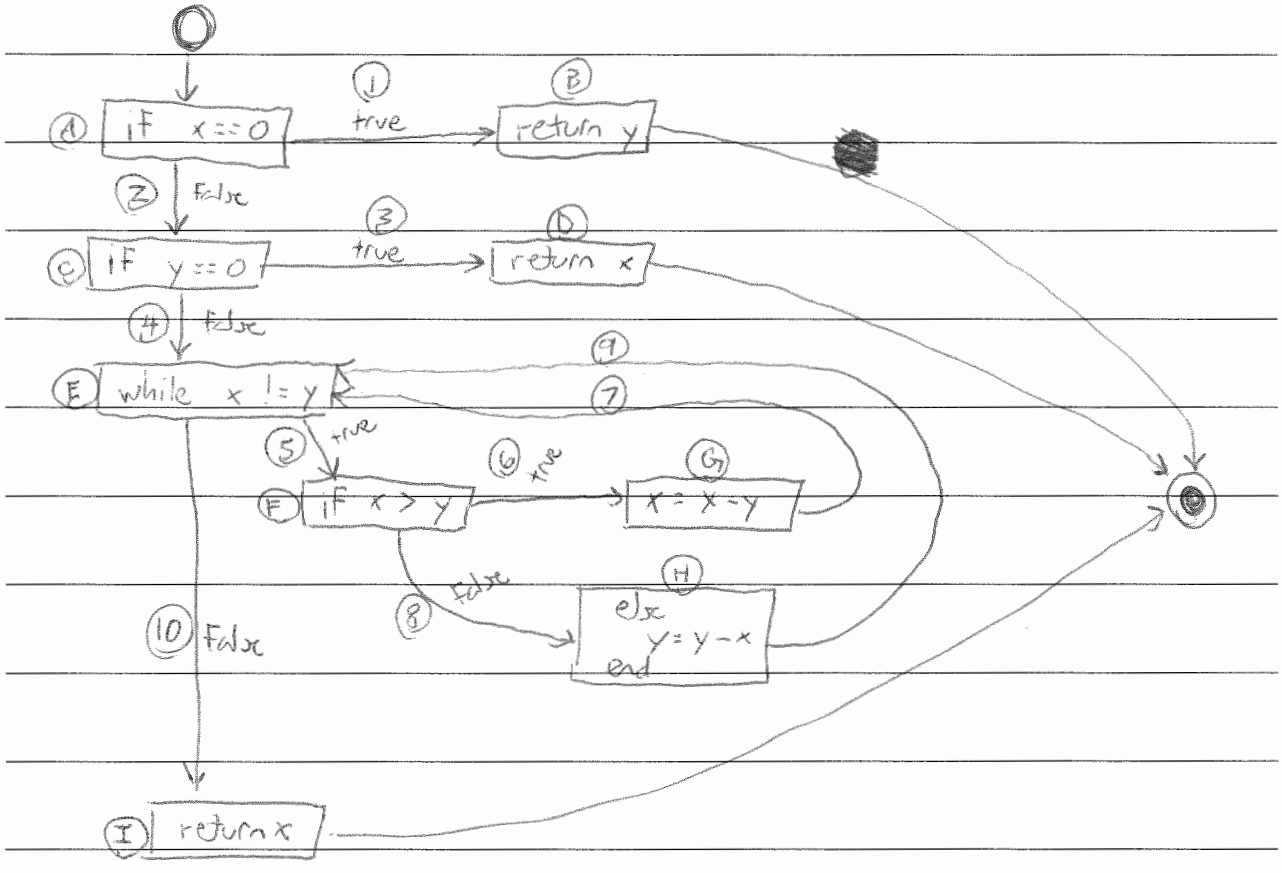
What change to a line in the function would introduce a bug that your above test suite does not catch?

return x → return y - x

Consider this function.

```
def gcd(x, y)
    if x == 0
        return y
    end
    if y == 0
        return x
    end
    while x != y
        if x > y
            x = x - y
        else
            y = y - x
        end
    end
    return x
end
```

Draw a control-flow graph for the function. In addition to the usual CFG features, label the nodes with capital letters (A, B, C, etc.), and label the edges with numbers (1, 2, 3, etc.).



Fill in the table below with a test suite that provides statement coverage. In the Covers column, list the letter labels (A, B, C, etc.) of the nodes covered by each test case.

Input		Expected Output	Covers
x	y		
0	1	1	A, B
1	0	1	A, C, D
3	2	1	A, C, E, F, G, E, F, H, I

The function is correct to the best of my knowledge.

What change to a line in the function would introduce a bug that your above test suite catches?

x == 0 → x >= 0

What change to a line in the function would introduce a bug that your above test suite does not catch?

y = y - x → y = x = 1

Fill in the table below with a test suite that provides branch coverage. In the Covers column, list the number labels (1, 2, 3, etc.) of the edges covered by each test case (only true/false edges needed).

Input		Expected Output	Covers
x	y		
0	1	1	1
1	0	1	2, 3
3	2	1	2, 4, 5, 6, [redacted] 5, 8, [redacted] 10

The function is correct to the best of my knowledge.

What change to a line in the function would introduce a bug that your above test suite catches?

x == 0 → x >= 0

What change to a line in the function would introduce a bug that your above test suite does not catch?

y = y - x → y = x = 1

Fill in the table below with a test suite that provides path coverage. In the Covers column, list the number labels (1, 2, 3, etc.) of the edges covered by each test case. You need only cover executions that involve at most 1 iteration of each loop (if there are any).

Input		Expected Output	Covers
x	y		
0	1	1	1
1	0	1	2, 3
1	1	1	2, 4, 10
2	1	1	2, 4, 5, 6, 7, 10
1	2	1	2, 4, 5, 8, 9, 10
			Paths - 1 - 2, 3 - 2, 4, 10 - 2, 4, 5, 6, 7, 10 - 2, 4, 5, 8, 9, 10

The function is correct to the best of my knowledge.

What change to a line in the function would introduce a bug that your above test suite catches?

x == 0 → x >= 0

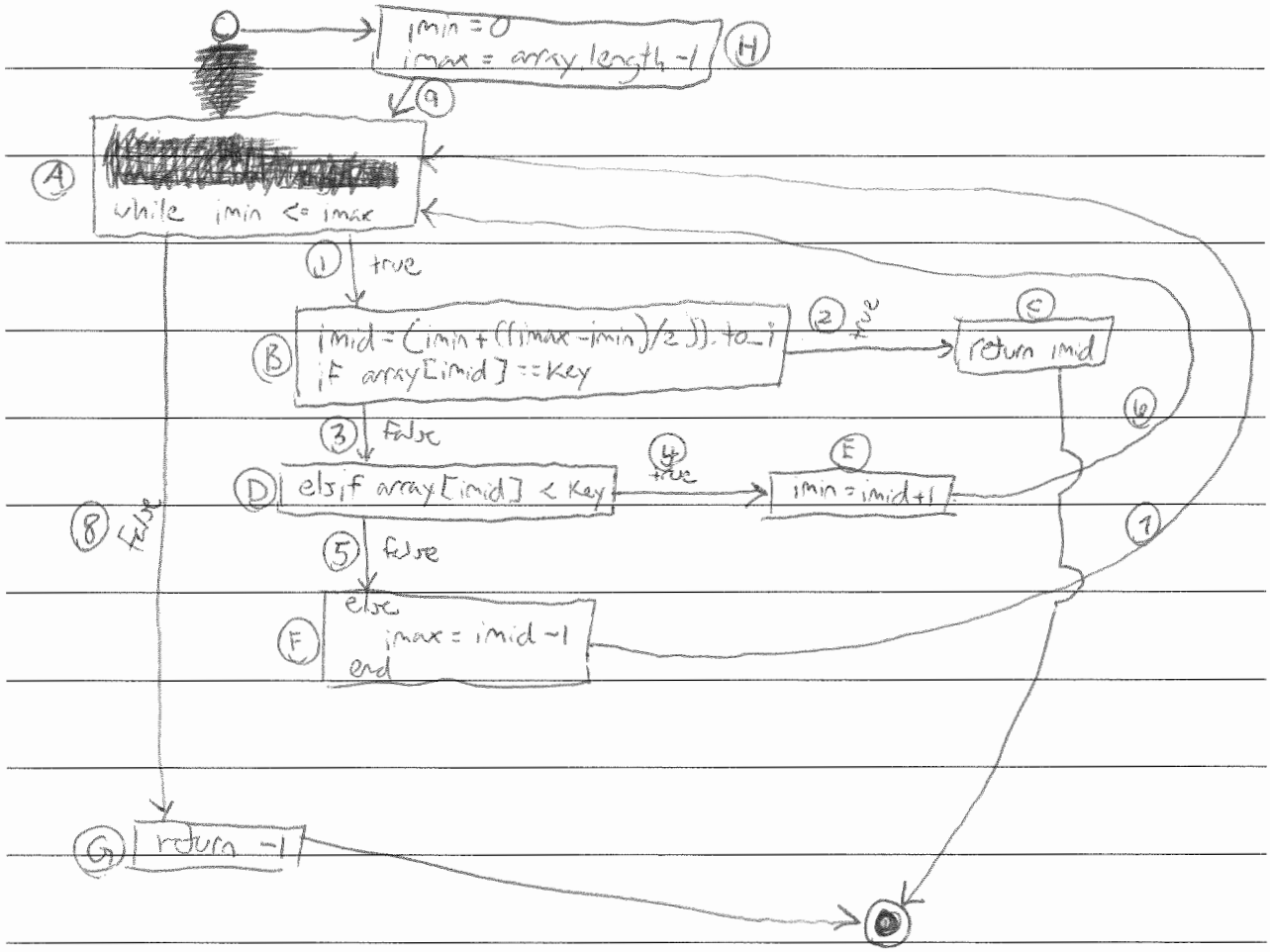
What change to a line in the function would introduce a bug that your above test suite does not catch?

return y → return 1

Consider this function.

```
def binary_search(array, key)
    imin = 0
    imax = array.length - 1
    while imin <= imax
        imid = (imin + ((imax - imin) / 2)).to_i
        if array[imid] == key
                        return imid
        elsif array[imid] < key
                        imin = imid + 1
        else
                        imax = imid - 1
        end
    end
    return -1
end
```

Draw a control-flow graph for the function. In addition to the usual CFG features, label the nodes with capital letters (A, B, C, etc.), and label the edges with numbers (1, 2, 3, etc.).



Fill in the table below with a test suite that provides statement coverage. In the Covers column, list the letter labels (A, B, C, etc.) of the nodes covered by each test case.

Input		Expected Output	Covers
array	key		
[ ]	a	-1	H, A, G
[a,b,c,d,e,f,g]	e	4	H, A, B, D, E, A, B, D, F, A, B, C

The function is correct to the best of my knowledge.

What change to a line in the function would introduce a bug that your above test suite catches?

return -1 → return 0

What change to a line in the function would introduce a bug that your above test suite does not catch?

return imid → return 4

Fill in the table below with a test suite that provides branch coverage. In the Covers column, list the number labels (1, 2, 3, etc.) of the edges covered by each test case (only true/false edges needed).

Input		Expected Output	Covers
array	key		
[ ]	a	-1	8
[a,b,c,d,e,f,g]	e	4	1, 3, 4, 1, 3, 5, 1, 2

The function is correct to the best of my knowledge.

What change to a line in the function would introduce a bug that your above test suite catches?

return -1 → return 0

What change to a line in the function would introduce a bug that your above test suite does not catch?

return mid → return 4

Fill in the table below with a test suite that provides path coverage. In the Covers column, list the number labels (1, 2, 3, etc.) of the edges covered by each test case. You need only cover executions that involve at most 1 iteration of each loop (if there are any).

Input		Expected Output	Covers
array	key		
[ ]	9	-1	9, 8
[9, 6, c]	6	1	9, 1, 2
[a]	6	-1	9, 1, 3, 4, 6, 8
[6]	a	-1	9, 1, 3, 5, 7, 8
			Paths – 9, 8 – 9, 1, 2
			– 9, 1, 3, 4, 6, 8 – 9, 1, 3, 5, 7, 8

The function is correct to the best of my knowledge.

What change to a line in the function would introduce a bug that your above test suite catches?

return -1 → return 0

What change to a line in the function would introduce a bug that your above test suite does not catch?

return imid → return 1