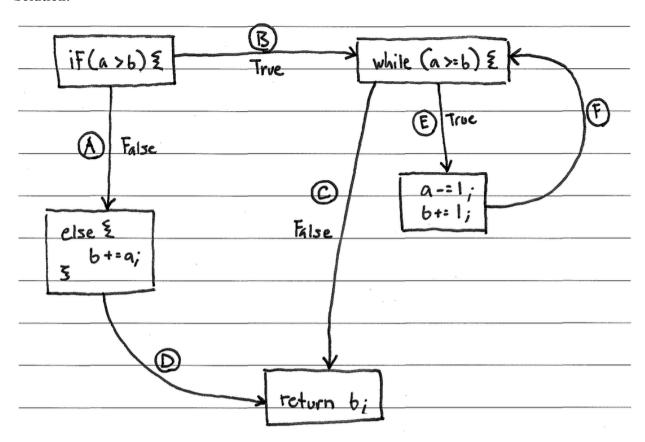
Problem: Draw a control flow diagram for this function. Label each edge with an uppercase letter.

```
int funWithNumbers(int a, int b) {
    if (a > b) {
        while (a >= b) {
            a -= 1;
            b += 1;
        }
    } else {
        b += a;
    }
    return b;
}
```



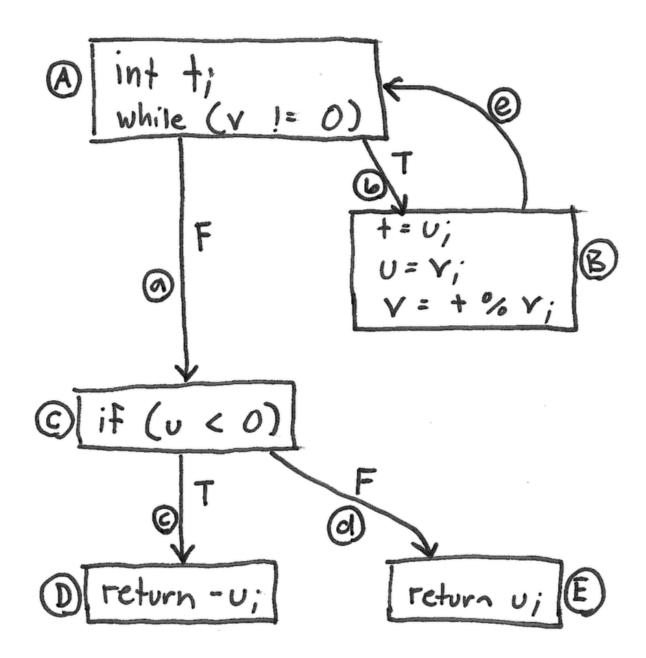
Problem: Fill in the table below with a test suite that provides *path coverage* of the code from the previous question. Cover no more than 2 iterations of the loop. In the covers column, list the relevant labeled items in your CFG that each test case covers. If there is some part of the coverage that is impossible to cover, then list it in the covers column, and put "N/A" in the associated x and y cells. Some cells in the table may be left blank.

Input		Covers	
X	y	Covers	

In	put	Covers
X	y	Covers
1	2	AD
N/A	N/A	BC.
1	0	BEFC
4	2	BEFEFC

Problem: Draw a control flow diagram for this function. Label each node in the graph with a capital letter, and label each edge with a lowercase letter.

```
int blammo(int u, int v) {
  int t;
  while (v != 0) {
    t = u;
    u = v;
    v = t % v; // Recall that % computes remainder of t/v
  }
  if (u < 0) { return -u; }
  return u;
}</pre>
```



Problems:

1. Fill in the table below with a test suite that provides <u>statement</u> coverage of the "blammo" code. In the covers column, list the relevant labeled items in your CFG that each test case covers. Some cells in the table may be left blank.

In	put	Covers		
u	V	Covers		

2. Fill in the table below with a test suite that provides <u>path</u> coverage of the "blammo" code. Cover no more than 1 iteration of the loop. In the covers column, list the relevant labeled items in your CFG that each test case covers. Some cells in the table may be left blank.

Input		Covers	
u	V	Covers	

1.

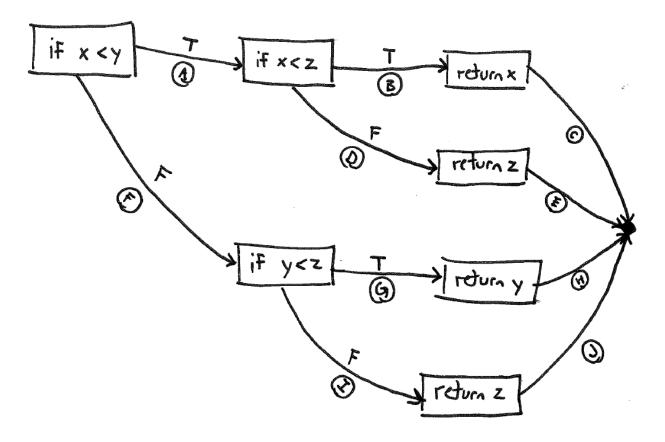
In	put	Covers	
u	V	Covers	
2	2	A, B, C, E	
-1	0	A, c, D	
		·	

2.

Inj	out	Covers	
u	V_	COVEIS	
-1	0	۵, ۵	
0	0	a, d	
-2	-2	b, e, a, c	
2	2	b.e, a, d	

Problem: Draw a control-flow graph for the following function. Label each edge in the graph with an uppercase letter.

```
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</pre>
```

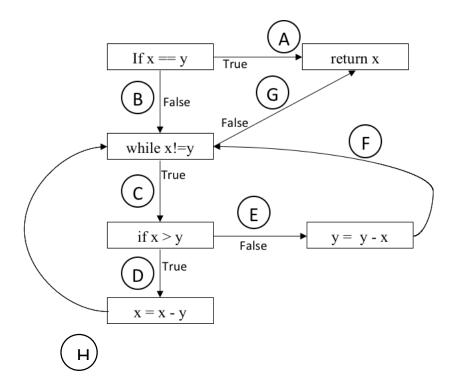


Problem: Fill in the table below with a test suite that provides <u>path coverage</u> of the min_of_three function from the previous question. In the covers column, list the relevant labeled edges in your CFG that each test case covers. Some cells in the table may be left blank.

Input		Expected Covers		
X	у	Z	Output	Covers

Input		Expected	Covers	
X	у	Z	Output	Covers
1	2	2		A, B, C
2	3	l	١	A, D, E
2		2	l	F, G, H
3	2	1		F, I, J

Consider the following control-flow graph for a gcd function in answering the questions below.



Problem: Fill in the table below with a test suite that provides <u>condition coverage</u> of the gcd function (see control-flow graph above). In the Covers column, list the relevant labeled edges in the CFG that each test case covers. Some cells in the table may be left blank.

Inj x	out V	Expected Output	Covers
	, <u>, , , , , , , , , , , , , , , , , , </u>		

Problem: Fill in the table below with a test suite that provides <u>path coverage</u> of the gcd function (see control-flow graph above). In the Covers column, list the relevant labeled edges in the CFG that each test case covers. Some cells in the table may be left blank. You need only cover executions that involve 1 iteration of the loop.

Input		Expected Output	Covers
X	у	Output	Covers

Solution: Condition Coverage

	Ing X	out y	Expected Output	Covers
	l	1	1	Α
	٢	2	١	B, C, E, G
alternative	(2	1	1	B, C, D, G
alter				
~	3	2		B, C, D, C, E, G

Solution: Path Coverage

Inj x	out	Expected Output	Covers
1	l	l	A
2	ı	1	B, C, D, H, G
1	2	1	B, C, E, F, G
			B, G — not possible

Consider this binary-search function and its associated control-flow graph.

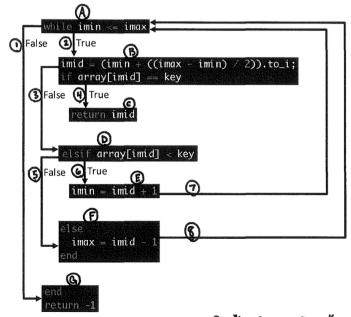
```
def binary_search(array, key, imin, imax)
  while imin <= imax
    imid = (imin + ((imax - imin) / 2)).to_i;
    if array[imid] == key
        return imid
    elsif array[imid] < key
        imin = imid + 1
        else
        imax = imid - 1
        end
    end
    return -1
end</pre>
```

Problems:

Consider the following test cases for the binary_search function.

	arra	ay		key	imin	imax
a.	[1]			0	0	0
b.	[1]			1	0	0
c.	[1]			1	1	0
d.	[1,	2,	3]	1	0	2
e.	[1,	2,	3]	2	0	2
f.	[1,	2,	3]	3	0	2
g.	[1,	2,	3]	1	2	0
h.	[1,	2,	3]	2	2	0
i.	[1,	2,	3]	3	2	0
					•	

1.	Select tests from the above to create a test suite that provides <u>statement</u> coverage of the <u>bina-ry_search</u> function. Your suite should contain the minimum number of tests to provide the coverage.
2.	Select tests from the above to create a test suite that provides <u>condition</u> coverage of the <u>bina-ry_search</u> function. Your suite should contain the minimum number of tests to provide the coverage.
3.	Select tests from the above to create a test suite that provides <u>path</u> coverage of the binary_search function. Cover only paths that contain one loop iteration or fewer (i.e., no path should enter the loop more than once). Your suite should contain the minimum number of tests to provide the coverage.



	array	Y		key	imin	imax	Statements Covered	Edges [®] Covered
a.	[1]			0	0	0	ABDFAG	23581
b.	[1]			1	0	0	ABC	24
c.	[1]			2	0	0	ABDEAG	23671
d.	[1, 2	2,	3]	1	0	2	ABDFABC	235824
e.	[1, 2	2,	3]	2	0	2	ABC	24
f.	[1, 2	2,	3]	3	0	2	ABDEABC	236724
g.	[1, 2	2,	3]	1	2	0	AG	ı
h.	[1, 2	2,	3]	2	2	0	AG	1
i.	[1, 2	2,	3]	3	2	0	AG	1

_____a, F or c,d ____(Need to cover statements A,B,C,D,E,F,G)

2.

a, F or c, d

(Need to cover edges 1, 2, 3, 4, 5, 6)

(9|h|i), (b|e), c, a

Rossible paths Tests that cover

Any 1 of Any 1 of

these these

2 3 581

C

a

Problems:

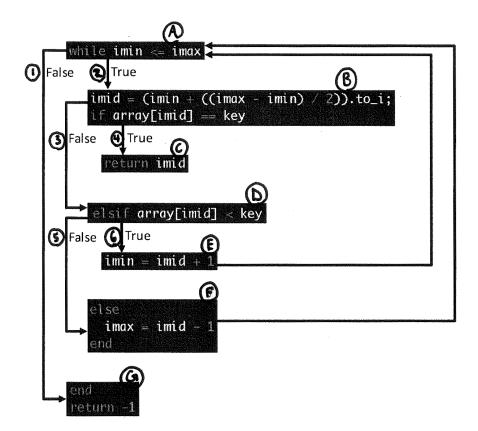
Consider the following test cases for the binary search function.

	array	key	imin	imax
a.	[0]	0	0	0
b.	[0]	1	0	0
c.	[0]	1	1	0
d.	[0]	-1	0	0

 Select tests from the above to create a test suite that provides <u>statement</u> coverage of the binary_search function. Your suite should contain the minimum number of tests to provide the coverage.

2. Select tests from the above to create a test suite that provides <u>condition</u> coverage of the <u>bina-ry_search</u> function. Your suite should contain the minimum number of tests to provide the coverage.

3. Select tests from the above to create a test suite that provides <u>path</u> coverage of the binary_search function. Cover only paths that contain one loop iteration or fewer (i.e., no path should enter the loop more than once). Your suite should contain the minimum number of tests to provide the coverage.



	array	key	imin	imax	Statements Coured	Condition Covered
a.	[0]	0	0	0	ABC	24
b.	[0]	1	0	0	ABDEG	2361
c.	[0]	1	1	0	A G	
d.	[0]	-1	0	0	ABDFG	2351

Pa	th		Walliago Chining			Cover	ત્ય	by	
A	G					c			
A	ß	C				19			
_A	B	D	E	A	G	16			
A	В	D	F	A	G	ld			

- 1. a, b, d
- 2. a, b, d
- 3. a, b, c, d

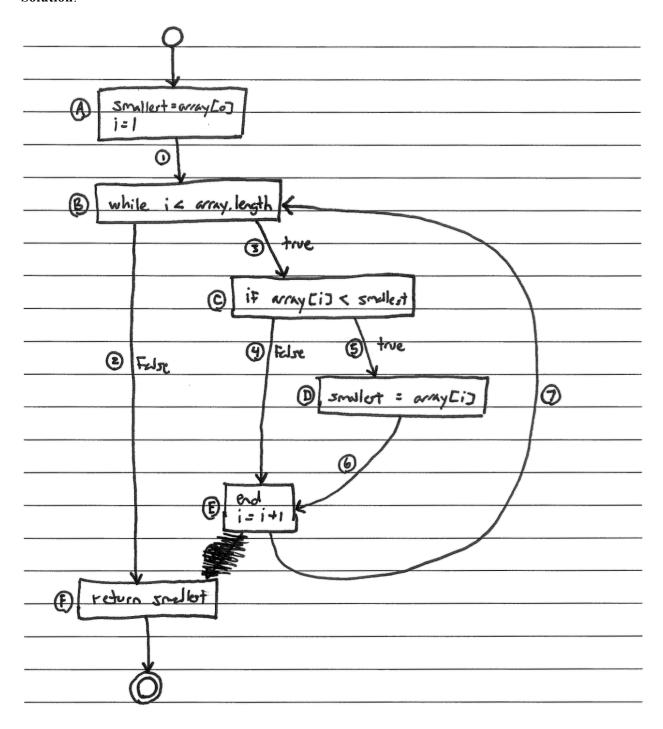
Consider this figure in answer the following questions.

```
def find_smallest(array)
  smallest = array[0]
  i = 1
  while i < array.length
    if array[i] < smallest
        smallest = array[i]
  end
  i = i + 1
  end
  return smallest
end</pre>
```

Figure 1. Function that finds the smallest value in an array.

Problem: Draw a control-flow graph (CFG) for the function in Figure 1. 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.). Don't forget to include entry and exit points.



Problems:

Use the CFG you created for the function in Figure 1 to answer the following questions.

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

Input array	Expected Output	Covers

2. Fill in the table below with a test suite that provides <u>branch coverage</u>. 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 array	Expected Output	Covers

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). Before you fill in the table, list all the paths to be covered. Paths: Expected Input Covers Output array

3. Fill in the table below with a test suite that provides path coverage. In the Covers column, list the

Multiple solutions are possible. These are just examples of correct solutions.

1.

Input array	Expected Output	Covers
[1,0]	0	A, B, C, D, E, B, F

2.

Input array	Expected Output	Covers
[1,0,2]	0	3, 3,5,4,2

3.

Input array	Expected Output	Covers
[o]	0	1,2
[1,0]	0	1,3,5,6,7,2
[0,1]	0	1, 3, 4, 7, 2