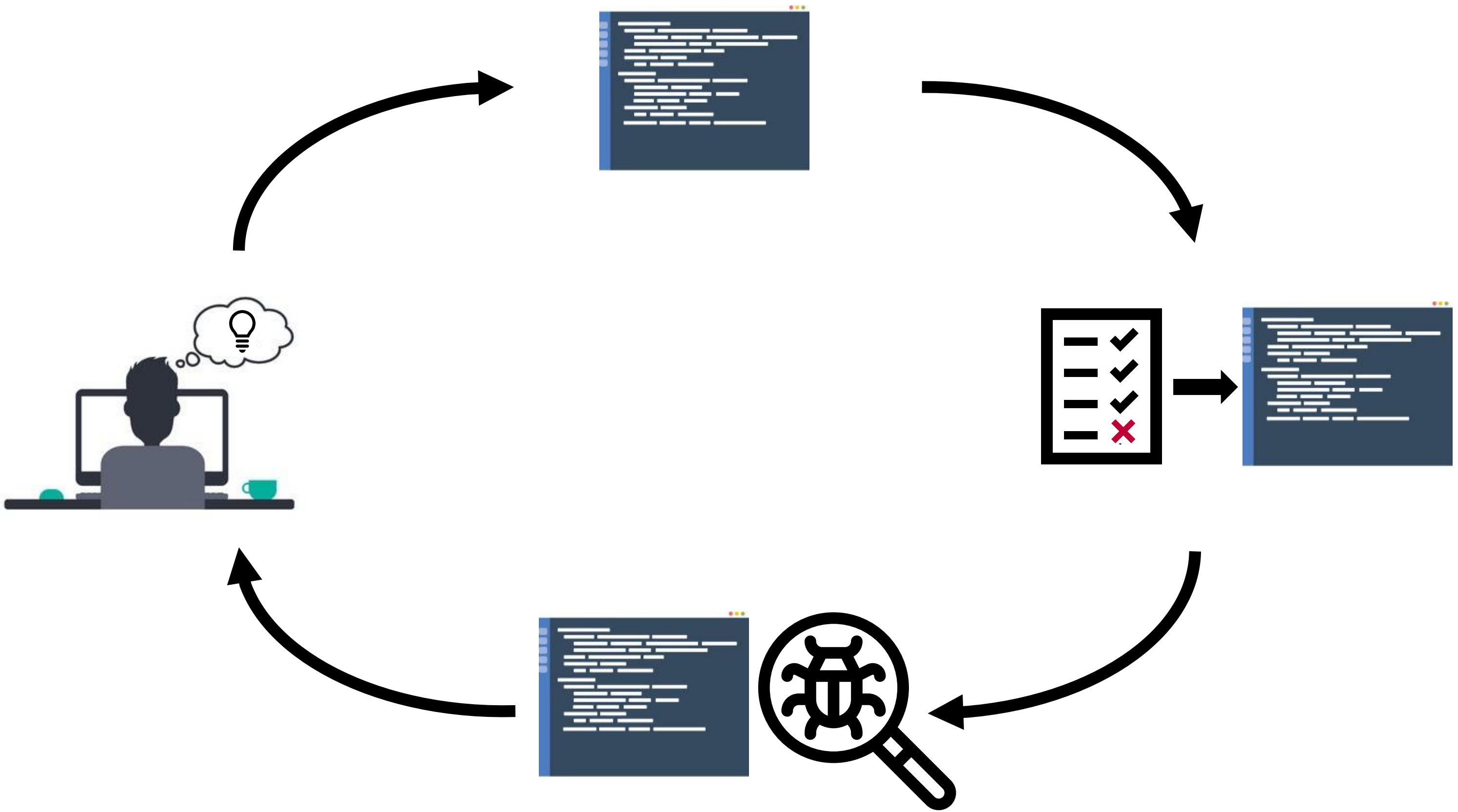
madPL

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

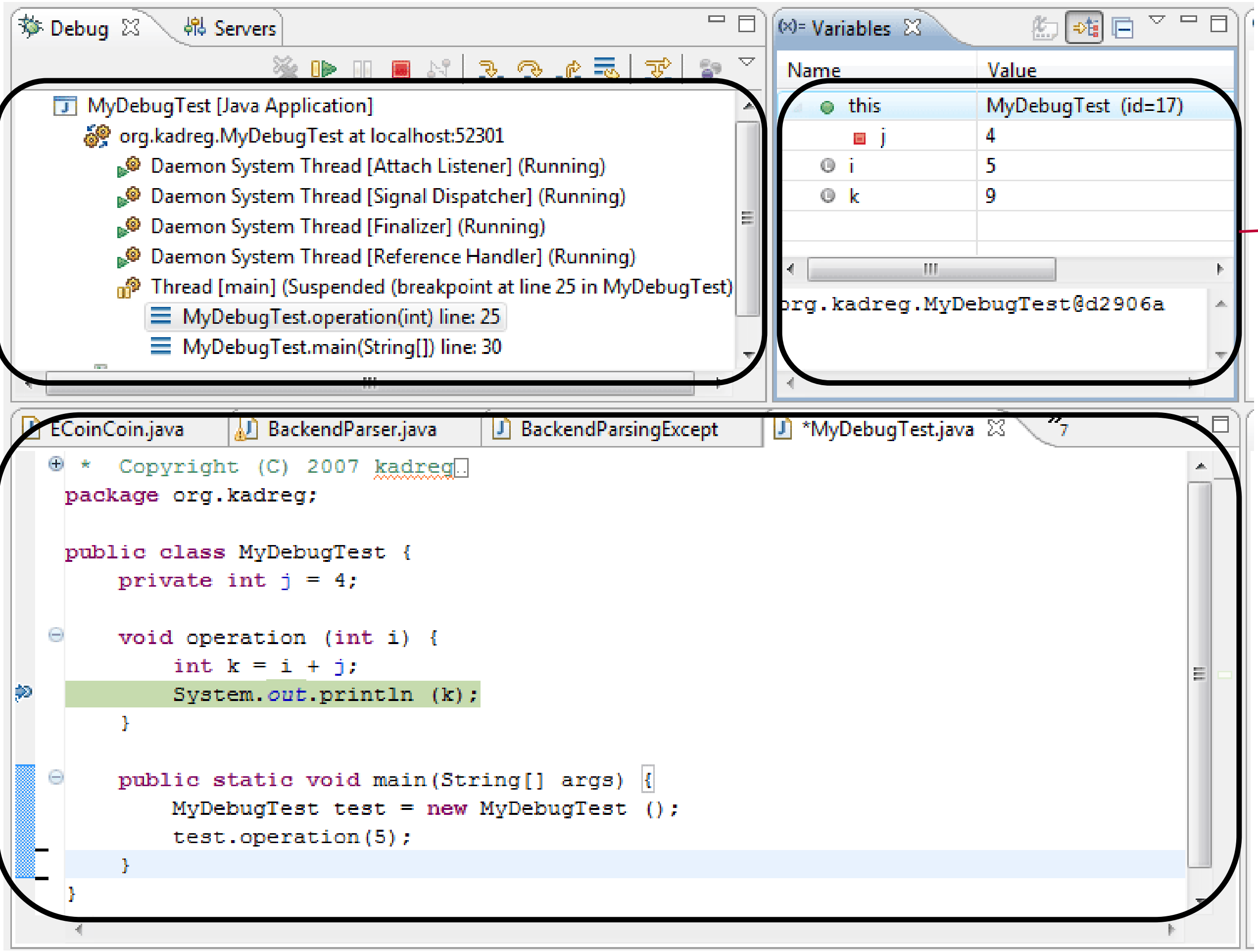
Direct Manipulation For Imperative Programs

Qinheping Hu[1] , Roopsha Samanta[2] , Rishabh Singh[3] , Loris D’Antoni[1]

[1] Universitve of Wisconsin-Madison
[2] Purdue University
[3] Google



Call stack



```
MyDebugTest [Java Application]
    org.kadreg.MyDebugTest at localhost:52301
        Daemon System Thread [Attach Listener] (Running)
        Daemon System Thread [Signal Dispatcher] (Running)
        Daemon System Thread [Finalizer] (Running)
        Daemon System Thread [Reference Handler] (Running)
        Thread [main] (Suspended (breakpoint at line 25 in MyDebugTest)
            MyDebugTest.operation(int) line: 25
            MyDebugTest.main(String[]) line: 30
```

Name	Value
this	MyDebugTest (id=17)
j	4
i	5
k	9

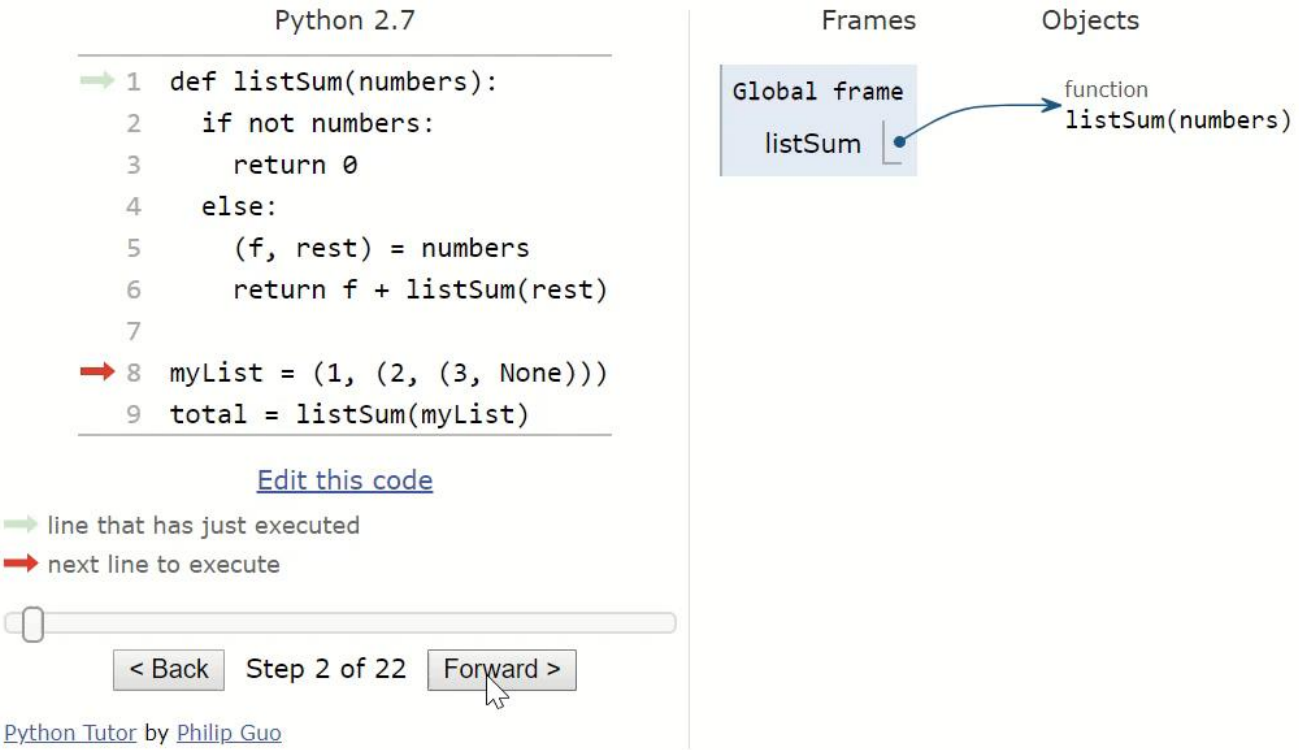
```
org.kadreg.MyDebugTest@d2906a
```

```
 * Copyright (C) 2007 kadreg
package org.kadreg;

public class MyDebugTest {
    private int j = 4;

    void operation (int i) {
        int k = i + j;
        System.out.println (k);
    }

    public static void main(String[] args) {
        MyDebugTest test = new MyDebugTest ();
        test.operation(5);
    }
}
```

Variable values

code

Python tutor [Guo SIGCSE 2013]

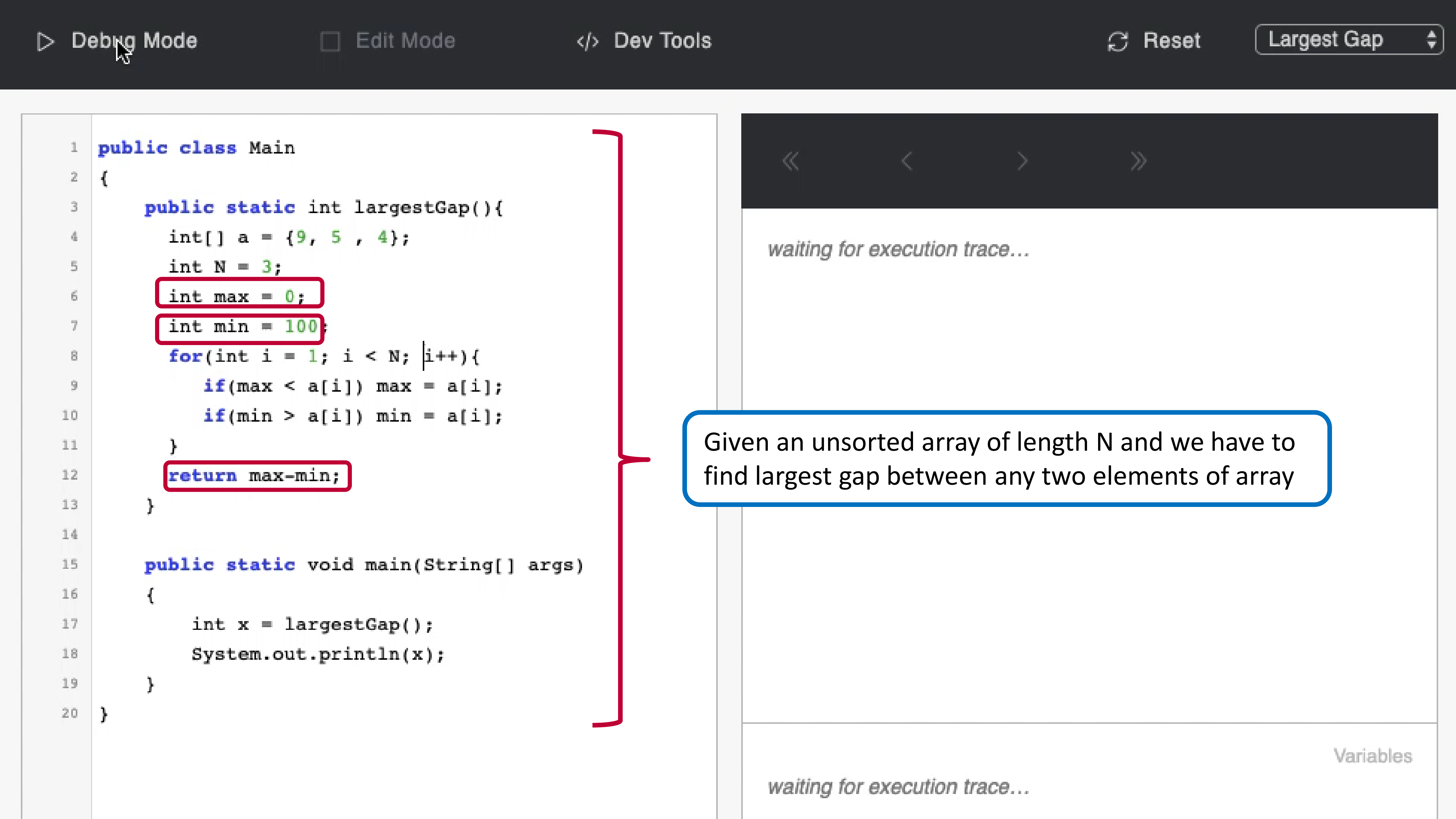


```
def listSum(numbers):
  if not numbers:
    return 0
  else:
    (f, rest) = numbers
    return f + listSum(rest)

myList = (1, (2, (3, None)))
total = listSum(myList)
```

→ line that has just executed

→ next line to execute

Step 2 of 22

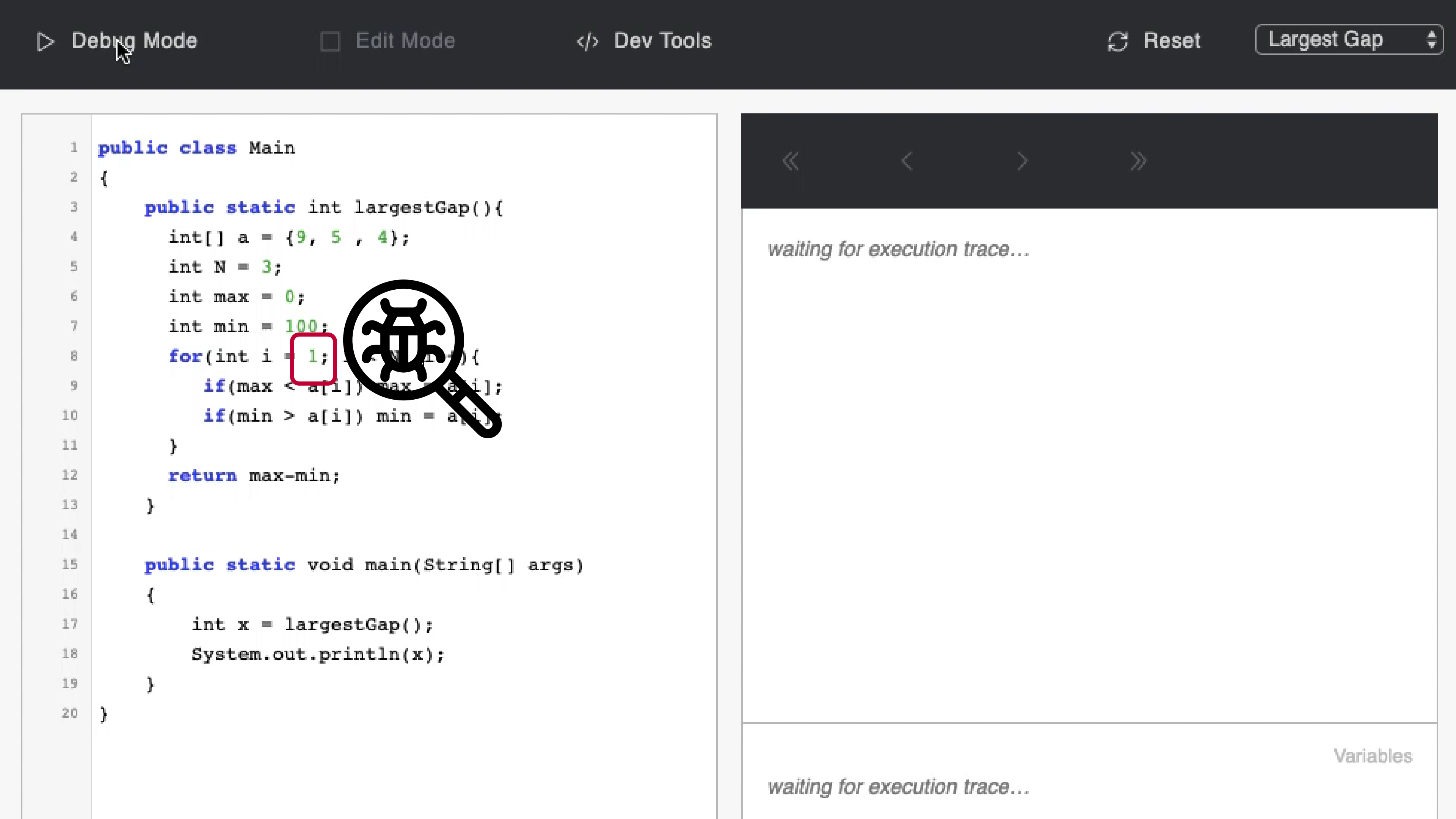


```
public class Main
{
  public static int largestGap(){
   int[] a = {9, 5 , 4};
   int N = 3;
   int max = 0;
   int min = 100;
   for(int i = 1; i < N; i++){
     if(max < a[i]) max = a[i];
     if(min > a[i]) min = a[i];
   }
   return max-min;
  }

  public static void main(String[] args)
  {
    int x = largestGap();
    System.out.println(x);
  }
}
```

waiting for execution trace…

waiting for execution trace…

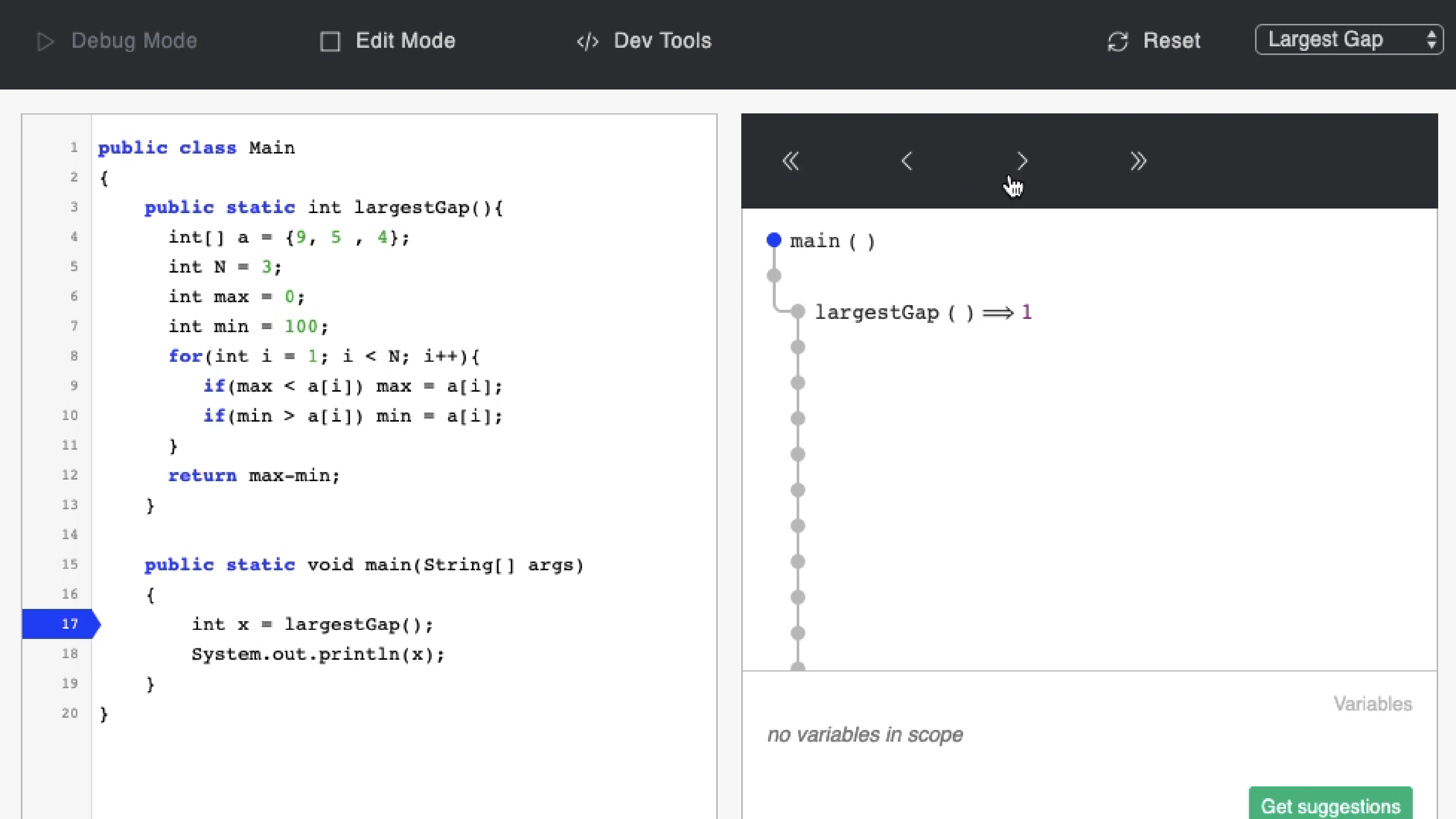
Given an unsorted array of length N and we have to find largest gap between any two elements of array



```java
public class Main
{
    public static int largestGap(){
      int[] a = {9, 5 , 4};
      int N = 3;
      int max = 0;
      int min = 100;
      for(int i = 1; i < N; i++){
         if(max < a[i]) max = a[i];
         if(min > a[i]) min = a[i];
      }
      return max-min;
    }

    public static void main(String[] args)
    {
        int x = largestGap();
        System.out.println(x);
    }
}
```

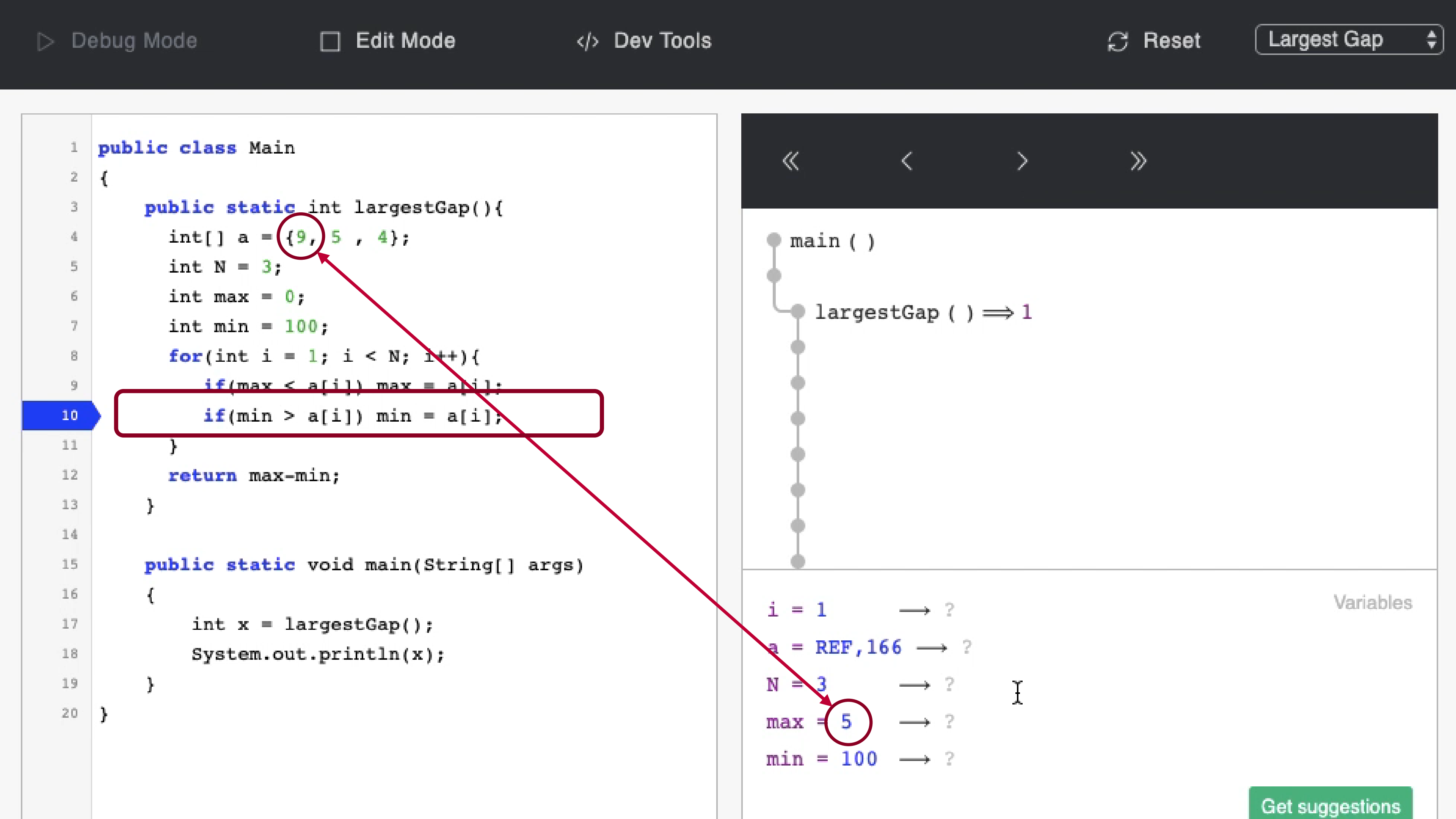
waiting for execution trace…

waiting for execution trace…



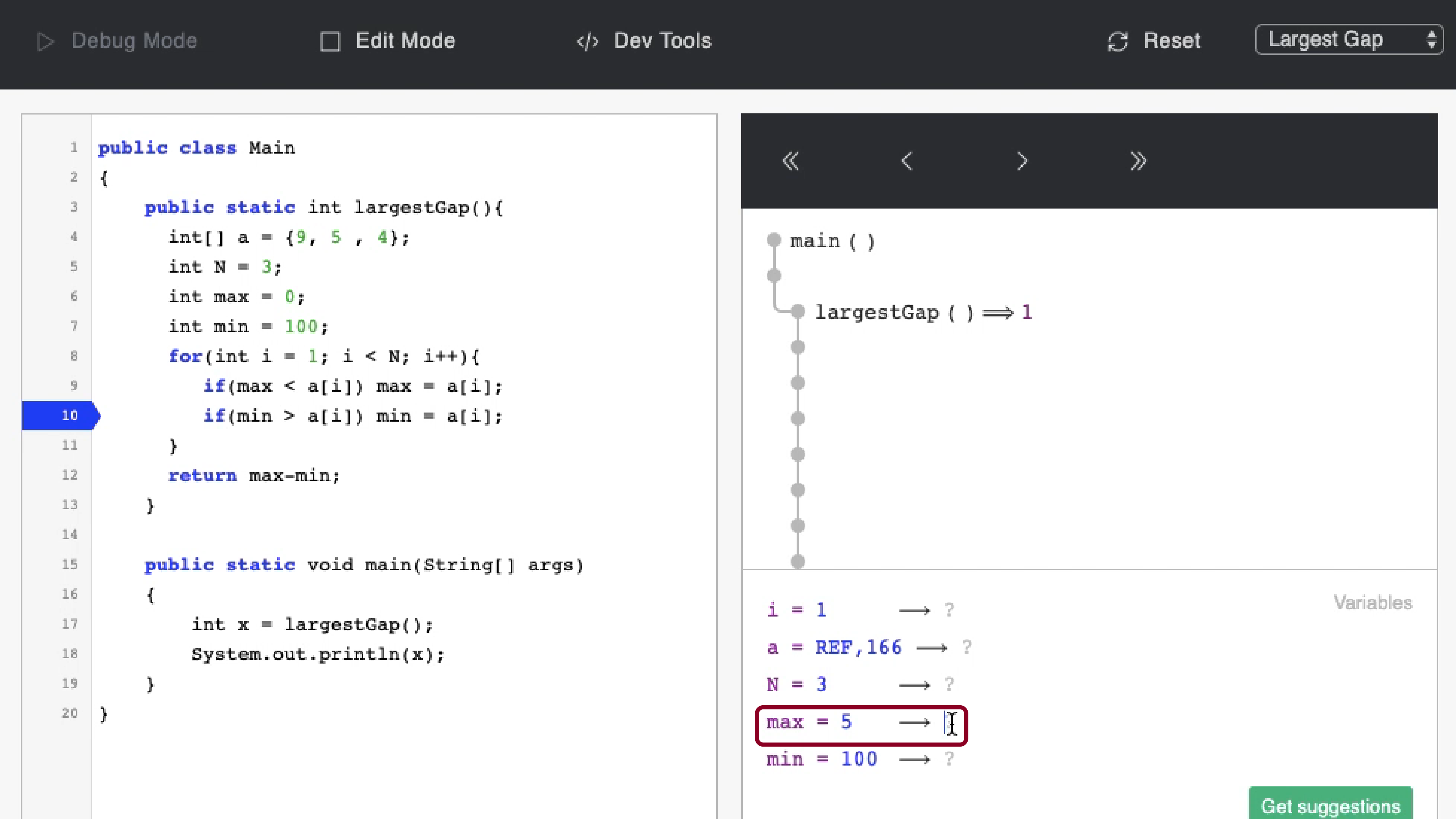
```
public class Main
{
    public static int largestGap(){
      int[] a = {9, 5 , 4};
      int N = 3;
      int max = 0;
      int min = 100;
      for(int i = 1; i < N; i++){
         if(max < a[i]) max = a[i];
         if(min > a[i]) min = a[i];
      }
      return max-min;
    }

    public static void main(String[] args)
    {
       int x = largestGap();
       System.out.println(x);
    }
}
```

no variables in scope



```java
public class Main
{
    public static int largestGap(){
      int[] a = {9, 5 , 4};
      int N = 3;
      int max = 0;
      int min = 100;
      for(int i = 1; i < N; i++){
        if(max < a[i]) max = a[i];
        if(min > a[i]) min = a[i];
      }
      return max-min;
    }

    public static void main(String[] args)
    {
        int x = largestGap();
        System.out.println(x);
    }
}
```

```
i = 1          ⟶ ?
a = REF,166    ⟶ ?
N = 3          ⟶ ?
max = 5        ⟶ ?
min = 100      ⟶ ?
```

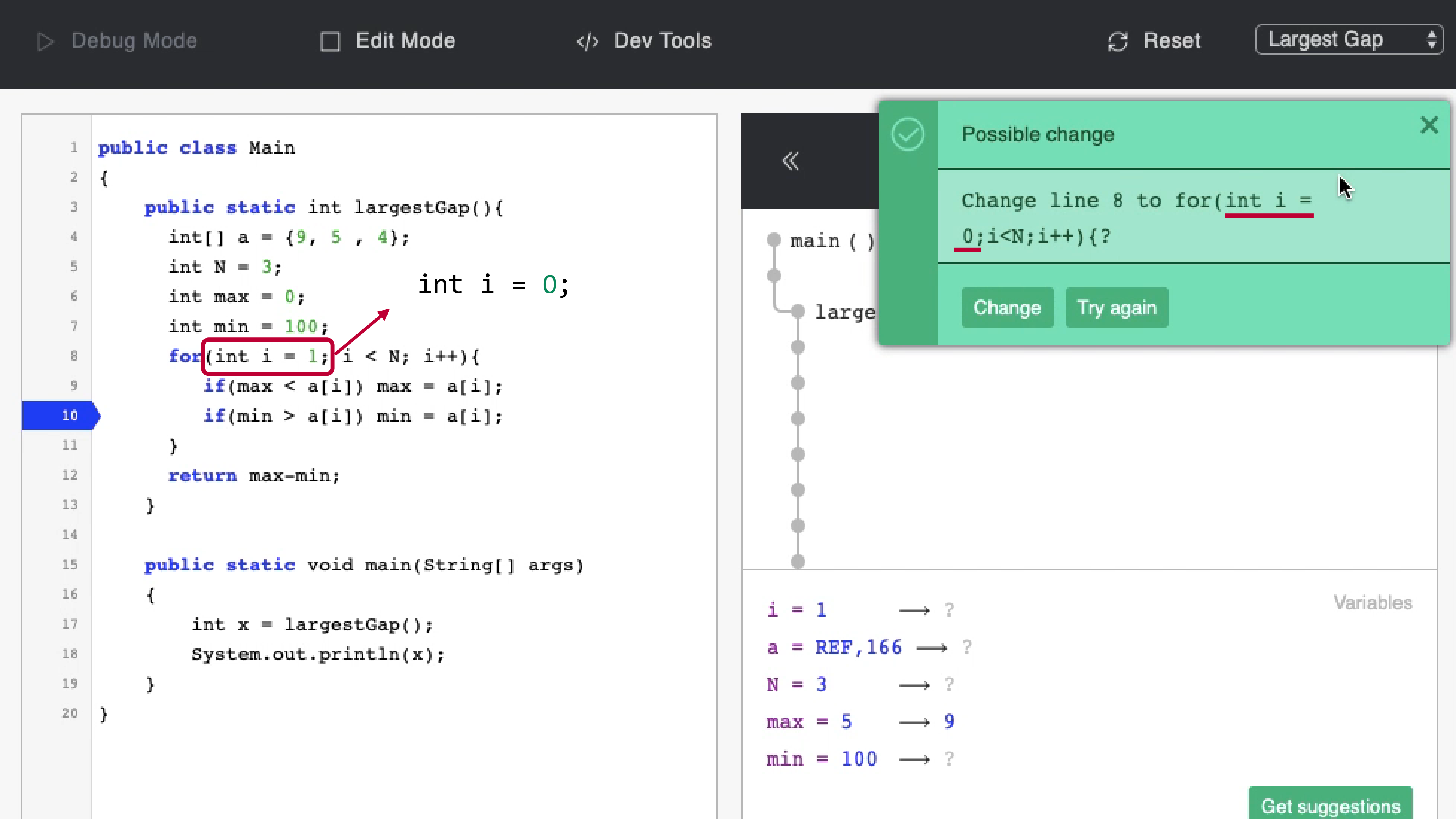


```java
public class Main
{
  public static int largestGap(){
   int[] a = {9, 5 , 4};
   int N = 3;
   int max = 0;
   int min = 100;
   for(int i = 1; i < N; i++){
     if(max < a[i]) max = a[i];
     if(min > a[i]) min = a[i];
   }
   return max-min;
  }

  public static void main(String[] args)
  {
    int x = largestGap();
    System.out.println(x);
  }
}
```

main ( )

largestGap ( ) ⟹ 1

```
i = 1       ⟶ ?
a = REF,166 ⟶ ?
N = 3       ⟶ ?
max = 5     ⟶
min = 100   ⟶ ?
```



```
1  public class Main
2  {
3      public static int largestGap(){
4        int[] a = {9, 5 , 4};
5        int N = 3;
6        int max = 0;
7        int min = 100;
8        for(int i = 1; i < N; i++){
9          if(max < a[i]) max = a[i];
10         if(min > a[i]) min = a[i];
11       }
12       return max-min;
13     }
14
15     public static void main(String[] args)
16     {
17         int x = largestGap();
18         System.out.println(x);
19     }
20 }
```

int i = 0;

Possible change

Change line 8 to for(int i = 0;i<N;i++){?

```
i = 1          ⟶ ?
a = REF,166    ⟶ ?
N = 3          ⟶ ?
max = 5        ⟶ 9
min = 100      ⟶ ?
```

Direct State Manipulation

Problem definition

Input

Buggy program

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 1;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i];
6       }
7   }
8   return max;
9 }
```

Direct manipulation

Manipulated location

Trace on `input = {9,5,6,10}`

loc	1	2	3	4	5	3
i	-	-	-	1	1	1
max	-	-	0	0	0	~~5~~ → 9

Goal

Repeired program

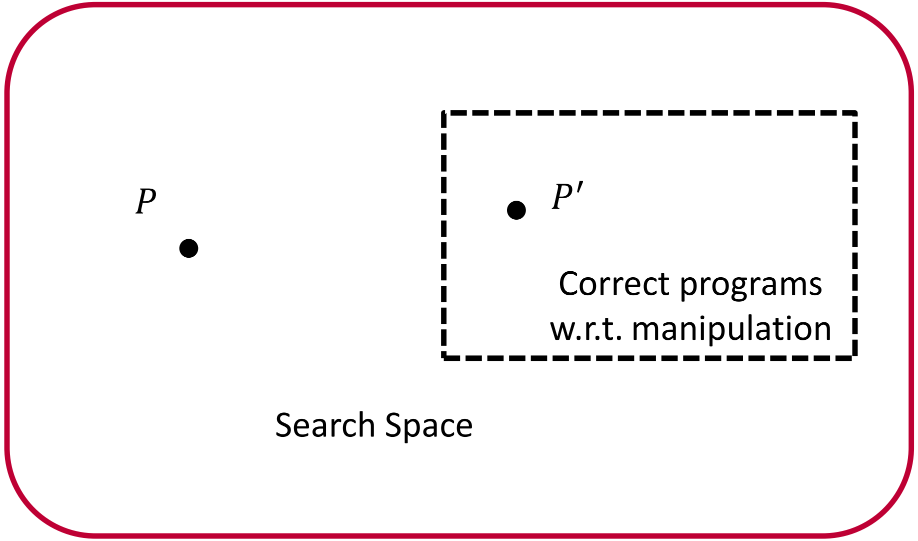
P'

New trace on `input = {9,5,6,10}`

loc	1	…	3	← Manipulat
i	-	…	?	← Don't car
max	-	…	9	← Manipulated value

P' is correct on the given manipulation

Problem definition



To solve this problem we need concrete ways to

- Describe the search space
- Specify the correctness
- Search for a solution

To solve this problem we need concrete ways to

- Describe the search space: program sketching
- Specify the correctness
- Search for a solution

How to describe the search space

```
public static int getMax(int[] input){
    int max = 0;
    for(int i = 1;i < input.length;i++){
        if(input[i] > max){
         max = input[i];
        }
    }
    return max;
}
```

```
public static int getMax(int[] input){
    int max = 0 + ??;
    for(int i = 1 + ??;i < input.length;i++){
        if(input[i] > max + ??){
         max = input[i] + ??;
        }
    }
    return max + ??;
}
```

Program Sketching [Solar-Lezama et al 06]

```
void P(int in){
    int c = ??;
    assert in + in == c * in;
}
```

```
void P(int in){
    int c = 2;
    assert in + in == c * in;
}
```

To solve this problem we need concrete ways to

- Describe the search space: program sketching
- Specify the correctness: guessing the return points
- Search for a solution

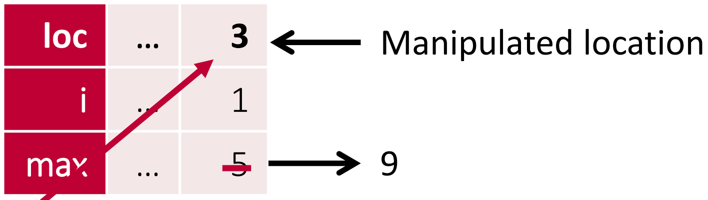
To solve this problem we need concrete ways to

- Describe the search space: program sketching
- Specify the correctness
- Search for a solution

Challenge 1: how to specify the manipulation

```
int pc = -1;
int[] trace_line;
public static int getMax(int[] input){
    record(2);
    int max = 0 + ??;
    record(3);
    for(int i = 1 + ??;i < input.length;i++){
        record(4);
        if(input[i] > max + ??){
            record(5);
            max = input[i] + ??;}
        record(3);
    }
    record(3);
    return max + ??;
}
assert ∃pc.trace_line[pc]==3;
```

Manipulation



loc	...	3
i	...	1
max	...	~~5~~

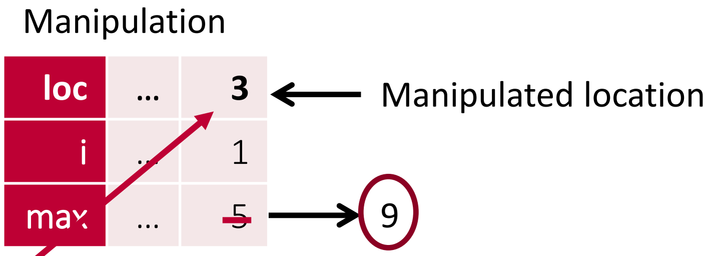
Manipulated location

9

```
void record(int line){
    pc++;
    trace_line[pc] = line;

}
```

Challenge 1: how to specify the manipulation

```
int pc = -1;
int[] trace_line;
public static int getMax(int[] input){
    record(2);
    int max = 0 + ??;
    record(3);
    for(int i = 1 + ??;i < input.length;i++){
        record(4);
        if(input[i] > max + ??){
            record(5);
            max = input[i] + ??;}
        record(3);
    }
    record(3);
    return max + ??;
}
assert ∃pc.trace_line[pc]==3;
```



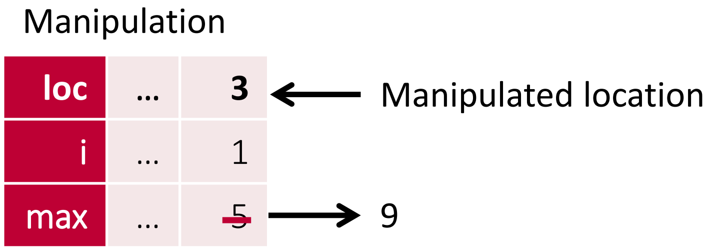
Manipulation

loc	…	3
i	…	1
max	…	~~5~~

```
void record(int line){
    pc++;
    trace_line[pc] = line;

}
```

Challenge 1: how to specify the manipulation

```
int pc = -1;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2);
    int max = 0 + ??;
    record(3);
    for(int i = 1 + ??;i < input.length;i++){
        record(4);
        if(input[i] > max + ??){
            record(5);
            max = input[i] + ??;}
        record(3);
    }
    record(3);
    return max + ??;
}
assert ∃pc.trace_line[pc]==3;
```



Manipulation

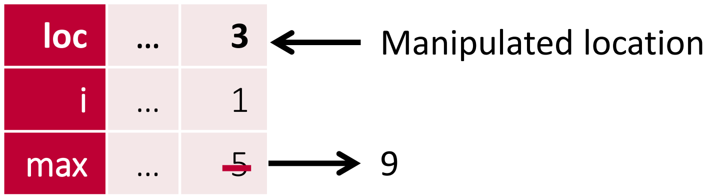
loc	…	**3**
i	…	1
max	…	~~5~~

```
void record(int line){
    pc++;
    trace_line[pc] = line;

}
```

Challenge 1: how to specify the manipulation

```
int pc = -1;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2,max);
    int max = 0 + ??;
    record(3,max);
    for(int i = 1 + ??;i < input.length;i++){
        record(4,max);
        if(input[i] > max + ??){
            record(5,max);
            max = input[i] + ??;}
        record(3,max);
    }
    record(3,max);
    return max + ??;
}
assert ∃pc.trace_line[pc]==3;
```

Manipulation



loc	…	**3**
i	…	1
max	…	~~5~~

Manipulated location

9

```
void record(int line){
    pc++;
    trace_line[pc] = line;

}
```

Challenge 1: how to specify the manipulation

```
int pc = -1;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2,max);
    int max = 0 + ??;
    record(3,max);
    for(int i = 1 + ??;i < input.length;i++){
        record(4,max);
        if(input[i] > max + ??){
            record(5,max);
            max = input[i] + ??;}
        record(3,max);
    }
    record(3,max);
    return max + ??;
}
assert ∃pc.trace_line[pc]==3;
```

Manipulation

loc	…	**3**	← Manipulated location
i	…	1	
max	…	~~5~~	→ 9

```
void record(int line, int max){
    pc++;
    trace_line[pc] = line;
    trace_max[pc] = max;
}
```

Challenge 1: how to specify the manipulation

```
int pc = -1;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2,max);
    int max = 0 + ??;
    record(3,max);
    for(int i = 1 + ??;i < input.length;i++){
        record(4,max);
        if(input[i] > max + ??){
            record(5,max);
            max = input[i] + ??;}
        record(3,max);
    }
    record(3,max);
    return max + ??;
}
assert ∃pc.trace_line[pc]==3 && trace_max[pc]==9;
```

Manipulation

loc	…	**3**	← Manipulated location
i	…	1	
max	…	~~5~~	→ 9

```
void record(int line, int max){
    pc++;
    trace_line[pc] = line;
    trace_max[pc] = max;
}
```

Challenge 2: at which iteration we should return

```
int pc = -1;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2,max);
    int max = 0 + ??;
    record(3,max);
    for(int i = 1 + ??;i < input.length;i++){
        record(4,max);
        if(input[i] > max + ??){
            record(5,max);
            max = input[i] + ??;}
        record(
    }
    record(3,max);
    return max + ??;
}
assert ∃pc.trace_line[pc]==3 && trace_max[pc]==9;
```

We want to find the repair instead of checking existence

Manipulation

loc	…	**3**
i	…	1
max	…	~~5~~ → 9

← Manipulated location

```
void record(int line, int max){
    pc++;
    trace_line[pc] = line;
    trace_max[pc] = max;
```

Challenge 2: at which iteration we should return

```
int pc = -1;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2,max);
    int max = 0 + ??;
    record(3,max);
    for(int i = 1 + ??;i < input.length;i++){
        record(4,max);
        if(input[i] > max + ??){
            record(5,max);
            max = input[i] + ??;}
        record(3,max);
    }
    record(3,max);
    return max + ??;
}
assert ∃pc.trace_line[pc]==3 && trace_max[pc]==9;
```

There can be multiple possible return points

Manipulation

loc	…	3	← Manipulated location
i	…	1	
max	…	~~5~~	→ 9

```
void record(int line, int max){
    pc++;
    e[pc] = line;
    trace_max[pc] = max;
}
```

Challenge 2: at which iteration we should return

```
int pc = -1, final_pc = ??;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2,max);
    int max = 0 + ??;
    record(3,max);
    for(int i = 1 + ??;i < input.length;i++){
        record(4,max);
        if(input[i] > max + ??){
            record(5,max);
            max = input[i] + ??;}
        record(3,max);
    }
    record(3,max);
    return max + ??;
}
assert ∃pc.trace_line[pc]==3 && trace_max[pc]==9;
```

Idea: guess the final program counter

loc	…	3	← Manipulated location
i	…	1	
max	…	~~5~~	→ 9

```
void record(int line, int max){
    pc++;
    trace_line[pc] = line;
    trace_max[pc] = max;
}
```

Challenge 2: at which iteration we should return

```
int pc = -1, final_pc = ??;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2,max);
    int max = 0 + ??;
    record(3,max); if(pc == final_pc) return;
    for(int i = 1 + ??;i < input.length;i++){
        record(4,max);
        if(input[i] > max + ??){
            record(5,max);
            max = input[i] + ??;}
        record(3,max);
    }
    record(3,max);
    return max + ??;
}
assert ∃pc.trace_line[pc]==3 && trace_max[pc]==9;
```

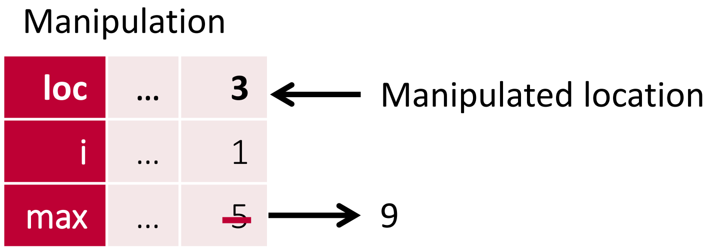
Manipulation

loc	…	**3**	← Manipulated location
i	…	1	
max	…	~~5~~	→ 9

```
void record(int line, int max){
    pc++;
    trace_line[pc] = line;
    trace_max[pc] = max;
}
```

Challenge 2: at which iteration we should return

```
int pc = -1, final_pc = ??;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2,max);
    int max = 0 + ??;
    record(3,max); if(pc == final_pc) return;
    for(int i = 1 + ??;i < input.length;i++){
        record(4,max);
        if(input[i] > max + ??){
            record(5,max);
            max = input[i] + ??;}
        record(3,max); if(pc == final_pc) return;
    }
    record(3,max);
    return max + ??;
}
assert ∃pc.trace_line[pc]==3 && trace_max[pc]==9;
```



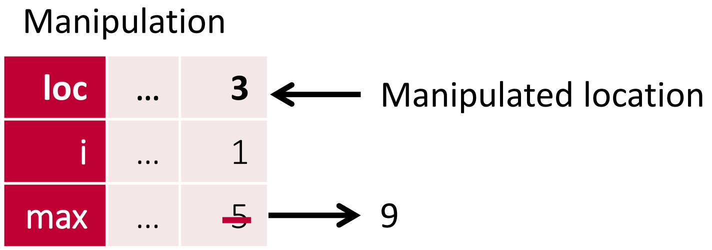
Manipulation

loc	…	**3**
i	…	1
max	…	~~5~~

```
void record(int line, int max){
    pc++;
    trace_line[pc] = line;
    trace_max[pc] = max;
}
```

Challenge 2: at which iteration we should return

```
int pc = -1, final_pc = ??;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2,max);
    int max = 0 + ??;
    record(3,max); if(pc == final_pc) return;
    for(int i = 1 + ??;i < input.length;i++){
        record(4,max);
        if(input[i] > max + ??){
            record(5,max);
            max = input[i] + ??;}
        record(3,max); if(pc == final_pc) return;
    }
    record(3,max); if(pc == final_pc) return;
    return max + ??;
}
assert ∃pc.trace_line[pc]==3 && trace_max[pc]==9;
```



Manipulation

loc	…	**3**	← Manipulated location
i	…	1	
max	…	~~5~~	→ 9

```
void record(int line, int max){
    pc++;
    trace_line[pc] = line;
    trace_max[pc] = max;
}
```

Challenge 2: at which iteration we should return

```
int pc = -1, final_pc = ??;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2,max);
    int max = 0 + ??;
    record(3,max); if(pc == final_pc) return;
    for(int i = 1 + ??;i < input.length;i++){
        record(4,max);
        if(input[i] > max + ??){
            record(5,max);
            max = input[i] + ??;}
        record(3,max); if(pc == final_pc) return;
    }
    record(3,max); if(pc == final_pc) return;
    return max + ??;
}
assert ~~∃pc~~.trace_line[final_pc]==3 && trace_max[final_pc]==9;
```

Manipulation

loc	…	**3**
i	…	1
max	…	~~5~~

← Manipulated location

→ 9

```
void record(int line, int max){
    pc++;
    trace_line[pc] = line;
    trace_max[pc] = max;
}
```

Challenge 2: at which iteration we should return

```
int pc = -1, final_pc = ??;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2,max);
    int max = 0 + ??;
    record(3,max); if(pc == final_pc) return;
    for(int i = 1 + ??;i < input.length;i++){
        record(4,max);
        if(input[i] > max + ??){
            record(5,max);
            max = input[i] + ??;}
        record(3,max); if(pc == final_pc) return;
    }
    record(3,max); if(pc == final_pc) return;
    return max + ??;
}
```

Manipulation

loc	…	**3**	← Manipulated location
i	…	1	
max	…	~~5~~	→ 9

```
void record(int line, int max){
    pc++;
    trace_line[pc] = line;
    trace_max[pc] = max;
}
```

```
assert ~~∃pc~~.trace_line[final_pc]==3 && trace_max[final_pc]==9;
```

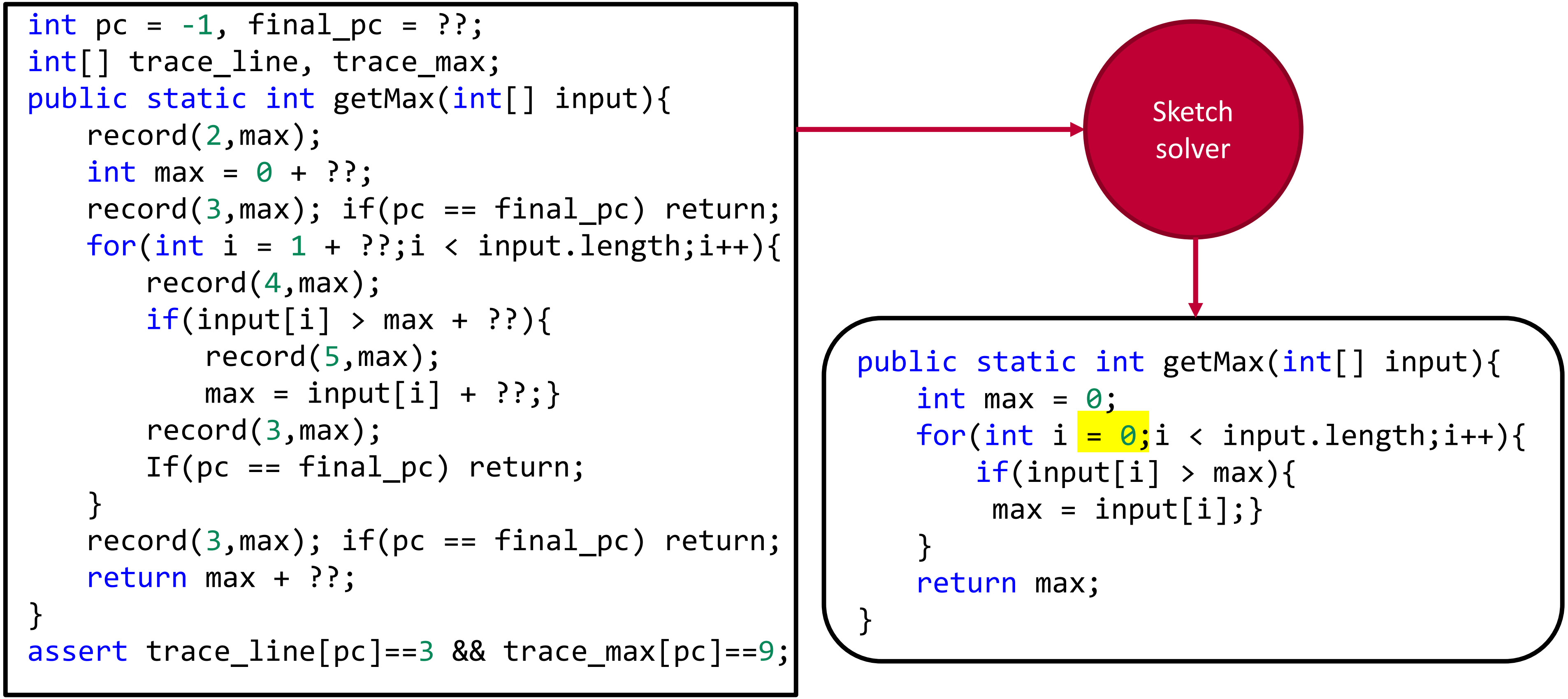
To solve this problem we need concrete ways to

- Describe the search space: program sketching
- Specify the correctness: guessing the return points
- Search for a solution:

To solve this problem we need concrete ways to

- Describe the search space: program sketching
- Specify the correctness: guessing the return points
- Search for a solution: Sketch solver

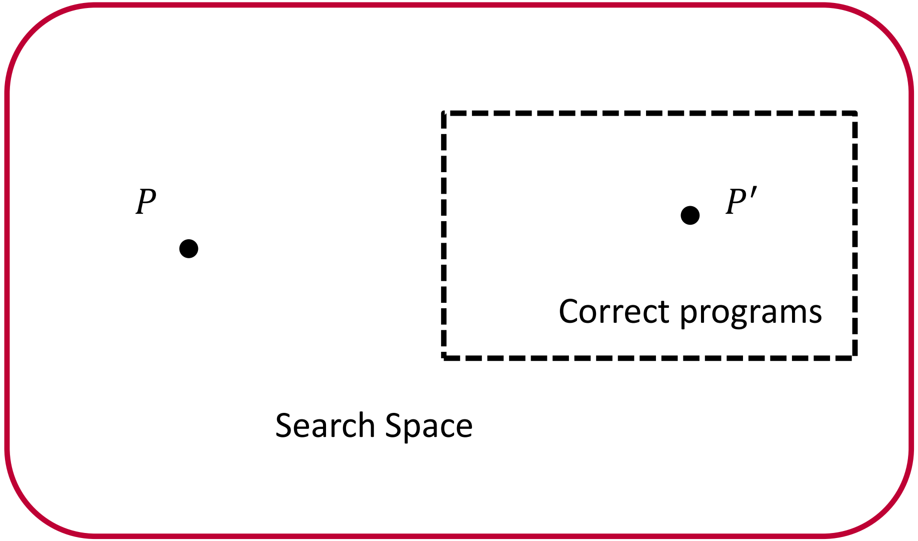
Find solution with Sketch solver



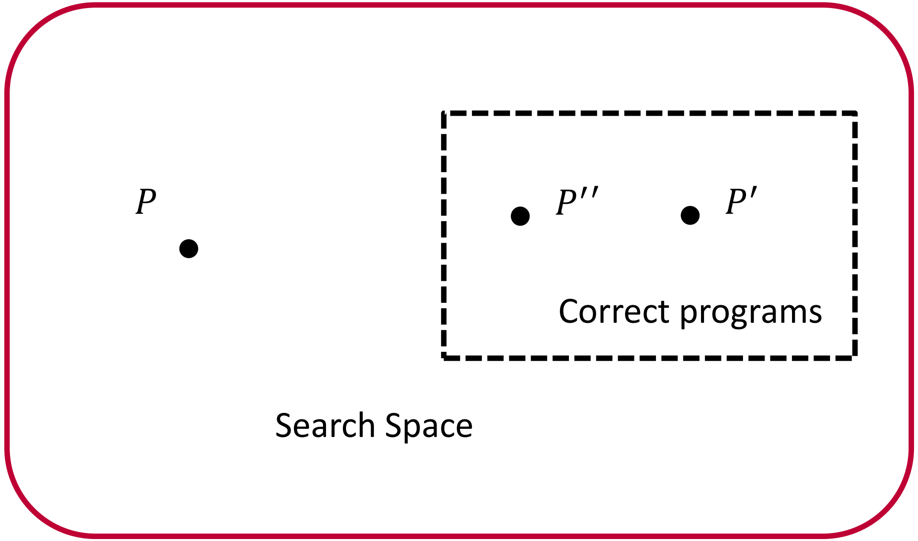
```
int pc = -1, final_pc = ??;
int[] trace_line, trace_max;
public static int getMax(int[] input){
    record(2,max);
    int max = 0 + ??;
    record(3,max); if(pc == final_pc) return;
    for(int i = 1 + ??;i < input.length;i++){
        record(4,max);
        if(input[i] > max + ??){
            record(5,max);
            max = input[i] + ??;}
        record(3,max);
        If(pc == final_pc) return;
    }
    record(3,max); if(pc == final_pc) return;
    return max + ??;
}
assert trace_line[pc]==3 && trace_max[pc]==9;
```

```
public static int getMax(int[] input){
    int max = 0;
    for(int i = 0;i < input.length;i++){
        if(input[i] > max){
         max = input[i];}
    }
    return max;
}
```

Finding a correct solution is not enough



When can we say a solution is better than another?



Idea 1: edit as less as possible

```
input = {9,5,6,10}
```

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 0;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i];}
6   }
7   return max;
8 }
```

Manipulation

loc	...	3
i	...	1
max	...	~~5~~ → 9

← Manipulated location

Change from 1 to 0: cost 1

Syntactic distance: syntactic similarity between programs

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 1;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i] + 4;}
6   }
7   return max;
8 }
```

Change from 0 to 4 : cost 4

Idea2: preserve the trace as much as possible

```
input = {9,5,6,10}
```

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 0;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i];}
6   }
7   return max;
8 }
```

Manipulation

loc	…	**3**	← Manipulated location
i	…	1	
max	…	~~5~~	→ 9

Idea2: preserve the trace as much as possible

```
input = {9,5,6,10}
```

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 0;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i];}
6   }
7   return max;
8 }
```

Change from 1 to 0

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 1;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i] - 1;}
6   }
7   return max;
8 }
```

Change from 0 to -1

Manipulation

loc	...	**3**
i	...	1
max	...	~~5~~

Manipulated location

9

Both edits are small

Idea2: preserve the trace as much as possible

```
input = {9,5,6,10}
```

```
1 public static int getMax(int[] input){
2   int ma
3   for(i                                       {
4       if
5        m
6   }
7   retur
8 }
```

New trace

loc	1	2	3	4	5	3
i	-	-	-	0	0	0
max	-	-	0	0	0	9

Original trace

loc	1	2	3	4	5	3
i	-	-	-	1	1	1
max	-	-	0	0	0	5

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 1;i < input.length;i++){
4      if(input[i] > max){
5       max = input[i] - 1;}
6   }
7   return max;
8 }
```

Idea2: preserve the trace as much as possible

```
input = {9,5,6,10}
```

New trace

loc	1	2	3	4	5	3
i	-	-	-	0	0	0
max	-	-	0	0	0	9

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 1;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i] - 1;}
6   }
7   return max;
8 }
```

Original trace

loc	1	2	3	4	5	3
i	-	-	-	1	1	1
max	-	-	0	0	0	5

Idea2: preserve the trace as much as possible

```
input = {9,5,6,10}
```

```
public static int getMax(int[] input){
  int ma
  for(i                                        {
      if
        m
  }
  return max;
}
```

New trace

loc	1	2	3	4	5	3
i	-	-	-	0	0	0
max	-	-	0	0	0	9

✓

Original trace

loc	1	2	3	4	5	3
i	-	-	-	1	1	1
max	-	-	0	0	0	5

```
public static int getMax(int[] input){
  int max = 0;
  }
  return max;
}
```

New trace

loc	1	2	3	4	5	3	4	5	3	4	5	3
i	-	-	-	1	1	1	2	2	2	3	3	3
max	-	-	0	0	0	4	4	4	5	5	5	9

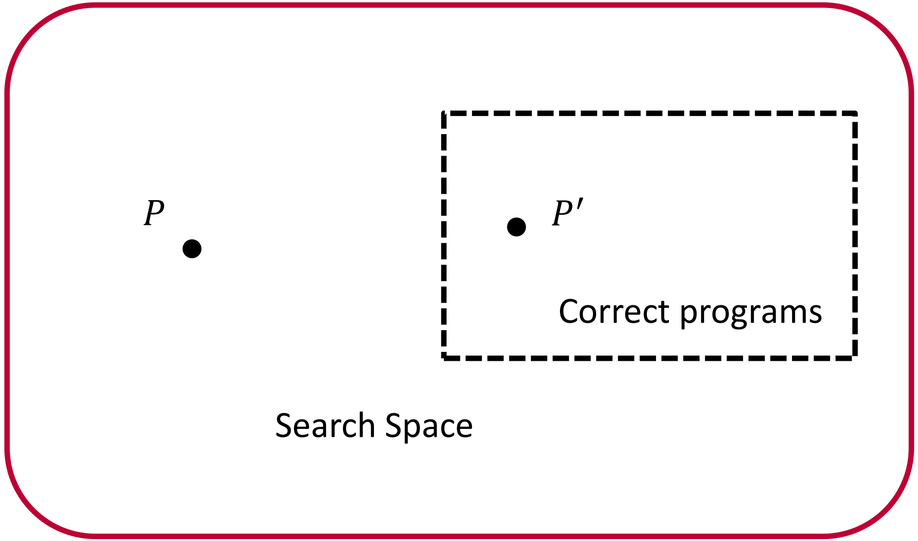
Semantic distance: similarity between traces

Program repair with Quantitative Objective [D’Antoni et al. CAV16]

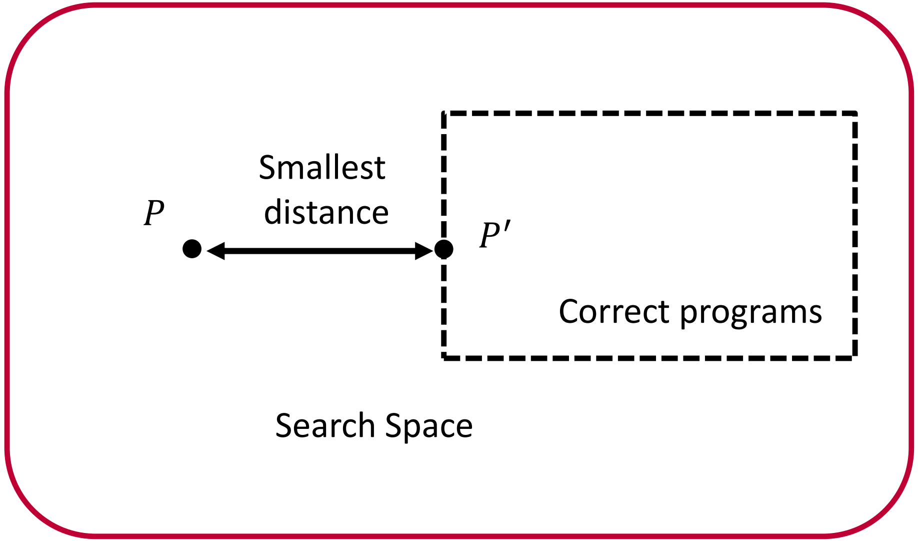
In program repair via test cases, finding solution with **smallest**

- **Syntactic distances:**
  - syntactic similarity between two programs
- **Semantic distances:**
  - similarity between trace of the original program and trace of repaired program on the given input

Revised problem definition with distance

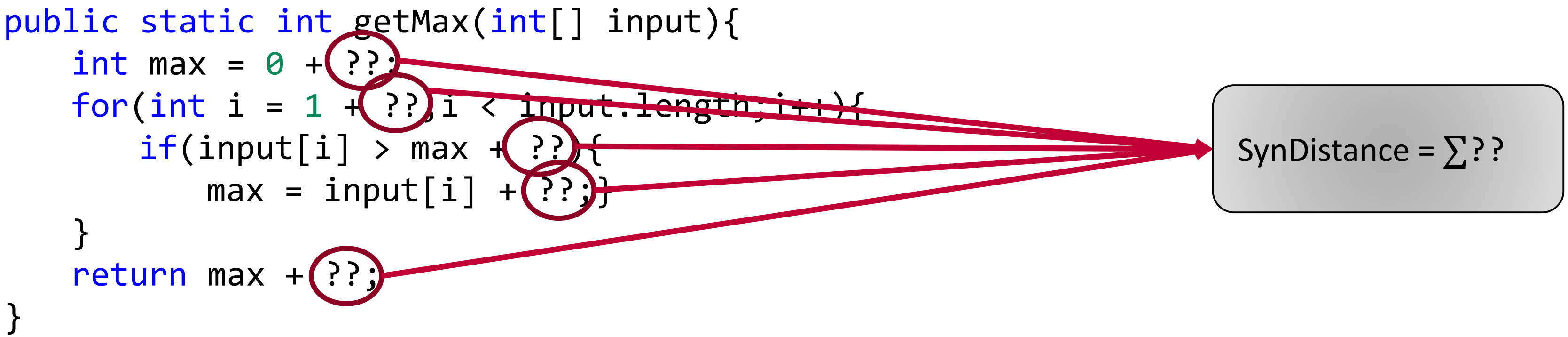


Revised problem definition with distance



Encode distance in Sketching

- Syntactic distance example: sum of values of all holes



```
public static int getMax(int[] input){
    int max = 0 + ??;
    for(int i = 1 + ??;i < input.length;i++){
        if(input[i] > max + ??){
            max = input[i] + ??;}
    }
    return max + ??;
}
assert trace_line[pc]==3 && trace_max[pc]==9;
```

SynDistance = $\sum ??$

Encode distance in Sketching

- Syntactic distance example: sum of values of all holes

```
public static int getMax(int[] input){
    int max = 0 + ??;
    for(int i = 1 + ??;i < input.length;i++){
        if(input[i] > max + ??){
            max = input[i] + ??;}
    }
    return max + ??;
}
assert trace_line[pc]==3 && trace_max[pc]==9;
```

```
int SyntacticDistance(){
    int dist = 0;
    for(int = 0; i < input_holes; i++){
        dist += ??_i
    }
    return dist;
}
```

Encode distance in Sketching

- Syntactic distance example: sum of values of all holes
- Semantic distance example: Hamming distance between traces

Original trace

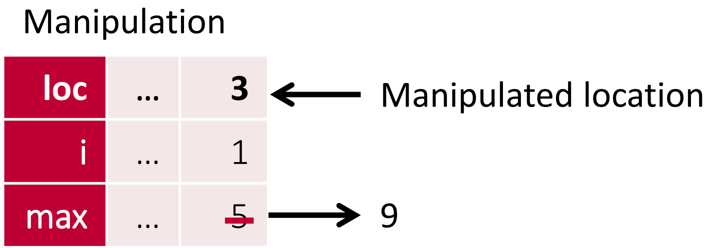
loc	1	2	3	4	5	3
i	-	-	-	1	1	1
max	-	-	0	0	0	5

New trace

loc	1	2	3	4	5	3
i	-	-	-	0	0	0
max	-	-	0	0	0	9

```
int SemanticDistance(int[] oriTrace, int[] trace){
    int dist = 0;
    for(int = 0; i < oriTrace.length; i++){
        dist += oriTrace[i] != trace[i];
    }
    return dist;
}
```

Quantitative objective in Sketch



Manipulation

loc	…	**3**
i	…	1
max	…	~~5~~

Manipulated location

9

```
assert trace_line[pc]==3 && trace_max[pc]==9;

minimize SyntacticDistance() + SemanticDistance(oriTrace,trace);
```

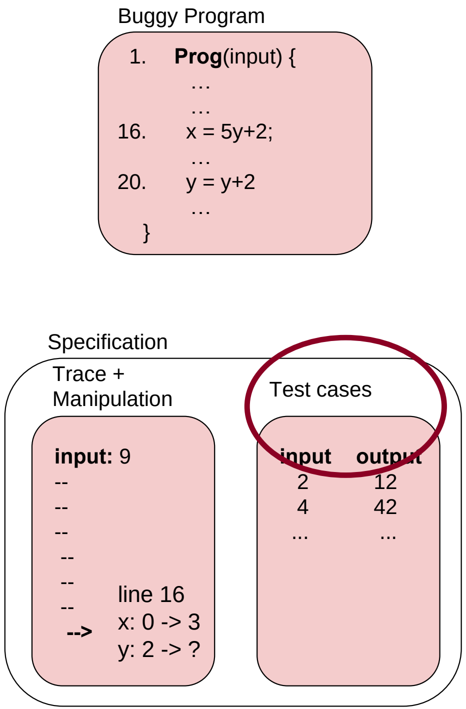
Overview of JDial

The tool JDial

Buggy Program

```
1.    Prog(input) {
        …
        …
16.    x = 5y+2;
        …
20.    y = y+2
        …
   }
```

The tool JDial



Buggy Program

```
1.    Prog(input) {
        …
        …
16.     x = 5y+2;
        …
20.     y = y+2
        …
   }
```

Specification

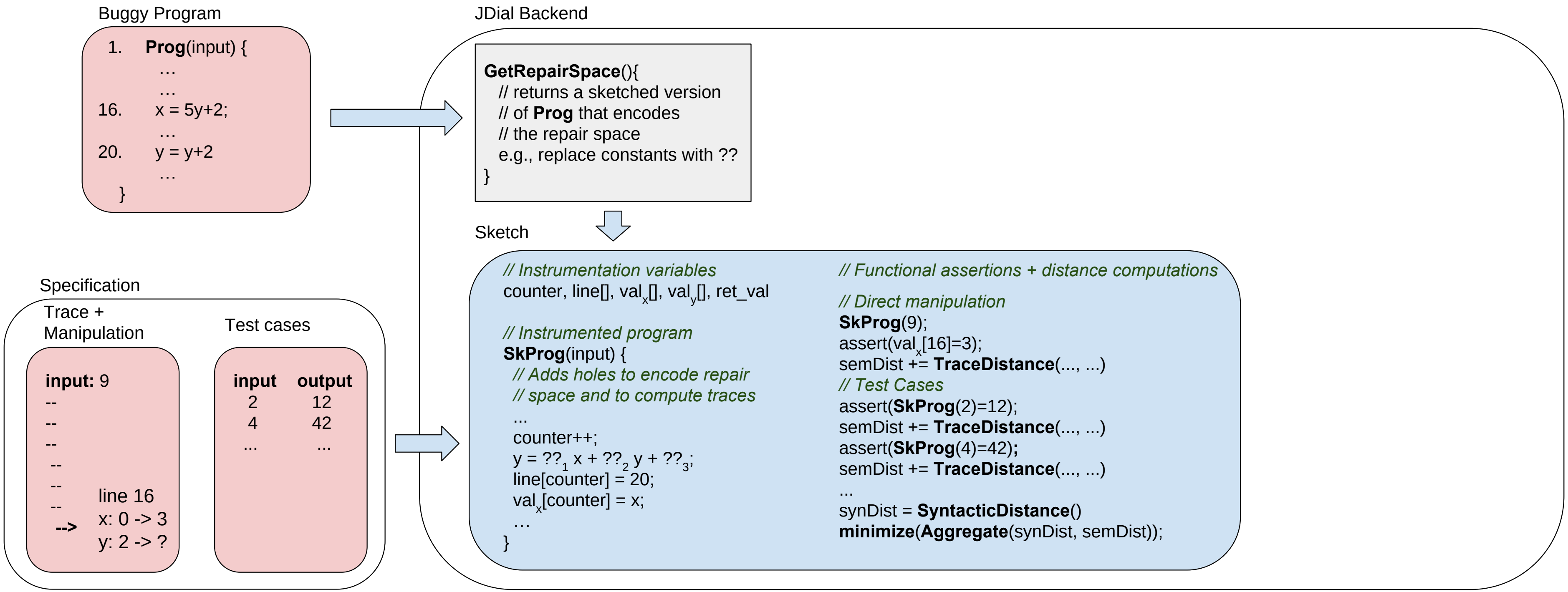
Trace + Manipulation

```
input: 9
--
--
--
 --
 --
 --        line 16
 -->       x: 0 -> 3
           y: 2 -> ?
```

Test cases

input	output
2	12
4	42
...	...

The tool JDial



Buggy Program

```
1.   Prog(input) {
        …
        …
16.     x = 5y+2;
        …
20.     y = y+2
        …
   }
```

Specification

Trace + Manipulation

```
input: 9
--
--
--
 --
 --
 --        line 16
  -->      x: 0 -> 3
           y: 2 -> ?
```

Test cases

input	output
2	12
4	42
...	...

JDial Backend

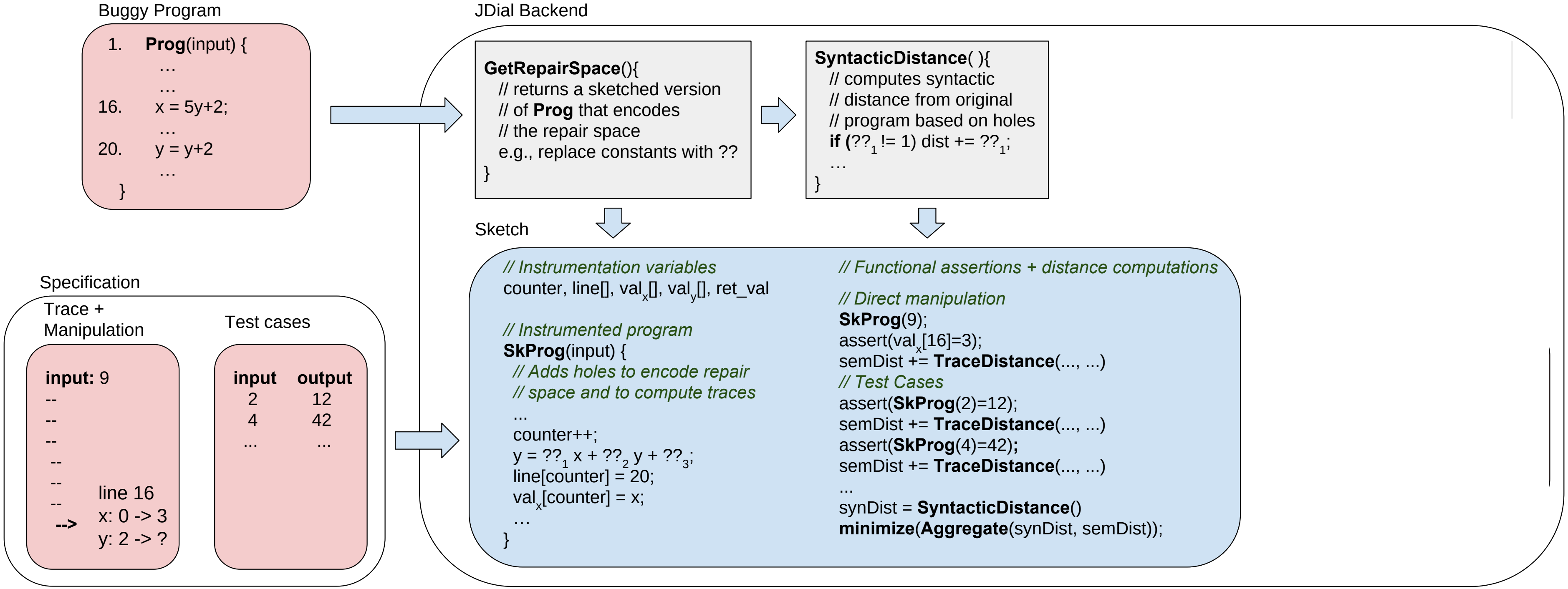
```
GetRepairSpace(){
  // returns a sketched version
  // of Prog that encodes
  // the repair space
  e.g., replace constants with ??
}
```

Sketch

```
// Instrumentation variables
counter, line[], val_x[], val_y[], ret_val

// Instrumented program
SkProg(input) {
  // Adds holes to encode repair
  // space and to compute traces
  ...
  counter++;
  y = ??_1 x + ??_2 y + ??_3;
  line[counter] = 20;
  val_x[counter] = x;
  …
}
```

```
// Functional assertions + distance computations

// Direct manipulation
SkProg(9);
assert(val_x[16]=3);
semDist += TraceDistance(..., ...)
// Test Cases
assert(SkProg(2)=12);
semDist += TraceDistance(..., ...)
assert(SkProg(4)=42);
semDist += TraceDistance(..., ...)
...
synDist = SyntacticDistance()
minimize(Aggregate(synDist, semDist));
```

The tool JDial



Buggy Program

```
1.   Prog(input) {
       …
       …
16.    x = 5y+2;
       …
20.    y = y+2
       …
   }
```

Specification

Trace + Manipulation

```
input: 9
--
--
--
 --
 --
 --          line 16
 -->         x: 0 -> 3
             y: 2 -> ?
```

Test cases

input	output
2	12
4	42
...	...

JDial Backend

```
GetRepairSpace(){
  // returns a sketched version
  // of Prog that encodes
  // the repair space
  e.g., replace constants with ??
}
```

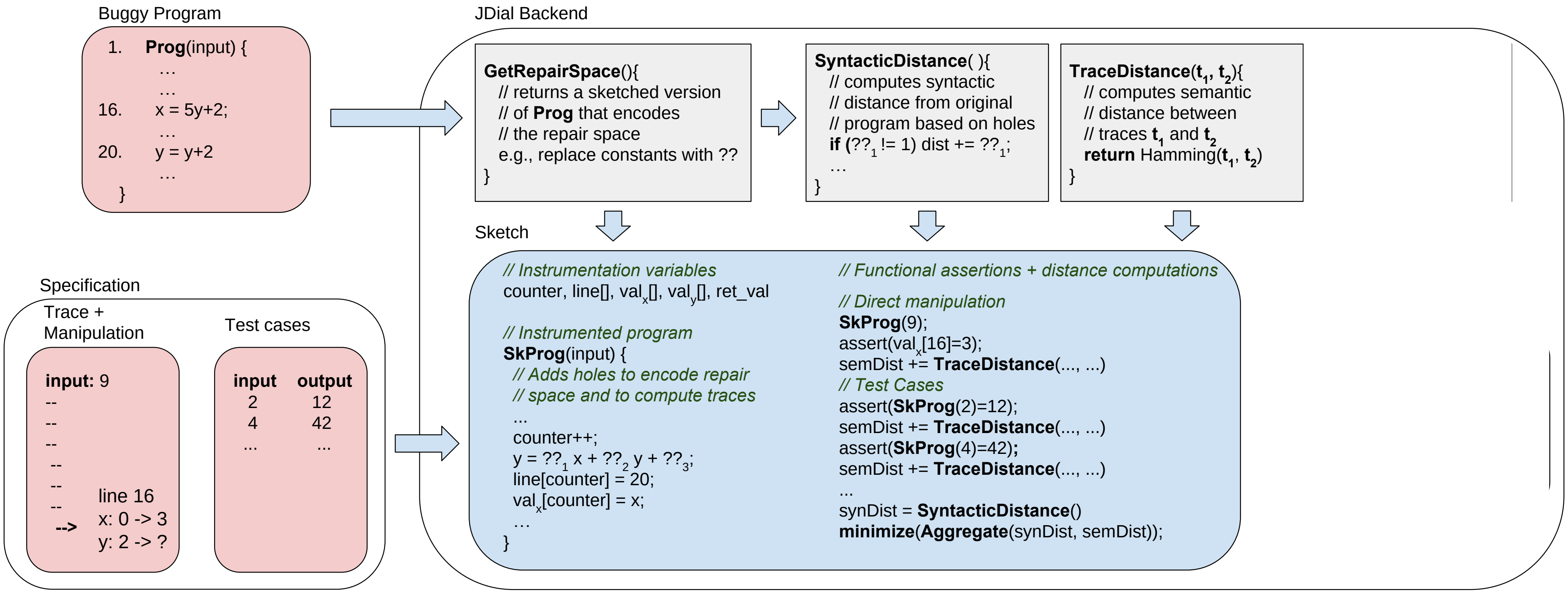
```
SyntacticDistance( ){
  // computes syntactic
  // distance from original
  // program based on holes
  if (??_1 != 1) dist += ??_1;
  …
}
```

Sketch

```
// Instrumentation variables
counter, line[], val_x[], val_y[], ret_val

// Instrumented program
SkProg(input) {
  // Adds holes to encode repair
  // space and to compute traces
  ...
  counter++;
  y = ??_1 x + ??_2 y + ??_3;
  line[counter] = 20;
  val_x[counter] = x;
  …
}
```

```
// Functional assertions + distance computations

// Direct manipulation
SkProg(9);
assert(val_x[16]=3);
semDist += TraceDistance(..., ...)
// Test Cases
assert(SkProg(2)=12);
semDist += TraceDistance(..., ...)
assert(SkProg(4)=42);
semDist += TraceDistance(..., ...)
...
synDist = SyntacticDistance()
minimize(Aggregate(synDist, semDist));
```

The tool JDial



Buggy Program

```
1.    Prog(input) {
        …
        …
16.     x = 5y+2;
        …
20.     y = y+2
        …
   }
```

Specification

Trace + Manipulation

```
input: 9
--
--
--
 --
 --
 --          line 16
 -->         x: 0 -> 3
             y: 2 -> ?
```

Test cases

input	output
2	12
4	42
...	...

JDial Backend

```
GetRepairSpace(){
  // returns a sketched version
  // of Prog that encodes
  // the repair space
  e.g., replace constants with ??
}
```

```
SyntacticDistance( ){
  // computes syntactic
  // distance from original
  // program based on holes
  if (??_1 != 1) dist += ??_1;
  …
}
```

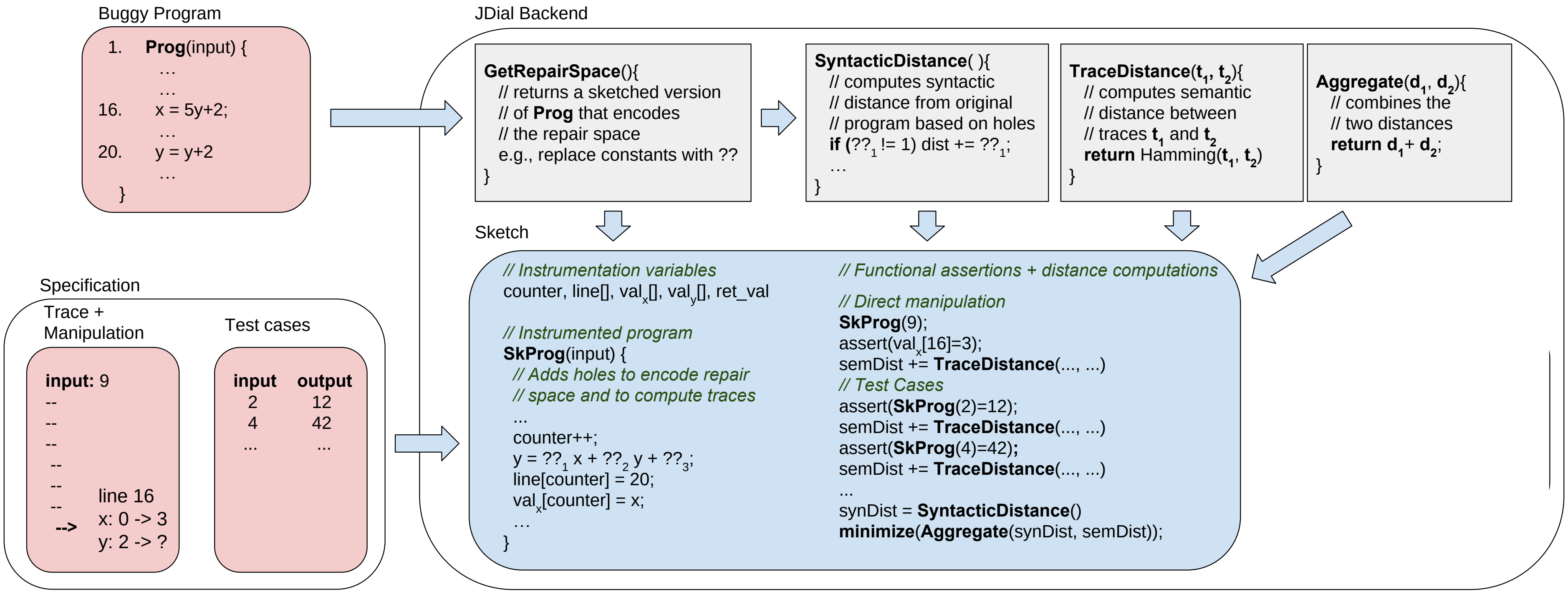
```
TraceDistance(t_1, t_2){
  // computes semantic
  // distance between
  // traces t_1 and t_2
  return Hamming(t_1, t_2)
}
```

Sketch

```
// Instrumentation variables
counter, line[], val_x[], val_y[], ret_val

// Instrumented program
SkProg(input) {
  // Adds holes to encode repair
  // space and to compute traces
  ...
  counter++;
  y = ??_1 x + ??_2 y + ??_3;
  line[counter] = 20;
  val_x[counter] = x;
  …
}
```

```
// Functional assertions + distance computations

// Direct manipulation
SkProg(9);
assert(val_x[16]=3);
semDist += TraceDistance(..., ...)
// Test Cases
assert(SkProg(2)=12);
semDist += TraceDistance(..., ...)
assert(SkProg(4)=42);
semDist += TraceDistance(..., ...)
...
synDist = SyntacticDistance()
minimize(Aggregate(synDist, semDist));
```

The tool JDial



Buggy Program

1. **Prog**(input) {
…
…
16. x = 5y+2;
…
20. y = y+2
…
}

Specification

Trace + Manipulation

input: 9
--
--
--
--
--
--
--> line 16
x: 0 -> 3
y: 2 -> ?

Test cases

input	**output**
2	12
4	42
…	…

JDial Backend

GetRepairSpace(){
// returns a sketched version
// of **Prog** that encodes
// the repair space
e.g., replace constants with ??
}

SyntacticDistance( ){
// computes syntactic
// distance from original
// program based on holes
if (??$_1$!= 1) dist += ??$_1$;
…
}

TraceDistance(**t$_1$**, **t$_2$**){
// computes semantic
// distance between
// traces **t$_1$** and **t$_2$**
return Hamming(**t$_1$**, **t$_2$**)
}

Aggregate(**d$_1$**, **d$_2$**){
// combines the
// two distances
return d$_1$+ **d$_2$**;
}

Sketch

// Instrumentation variables
counter, line[], val$_x$[], val$_y$[], ret_val

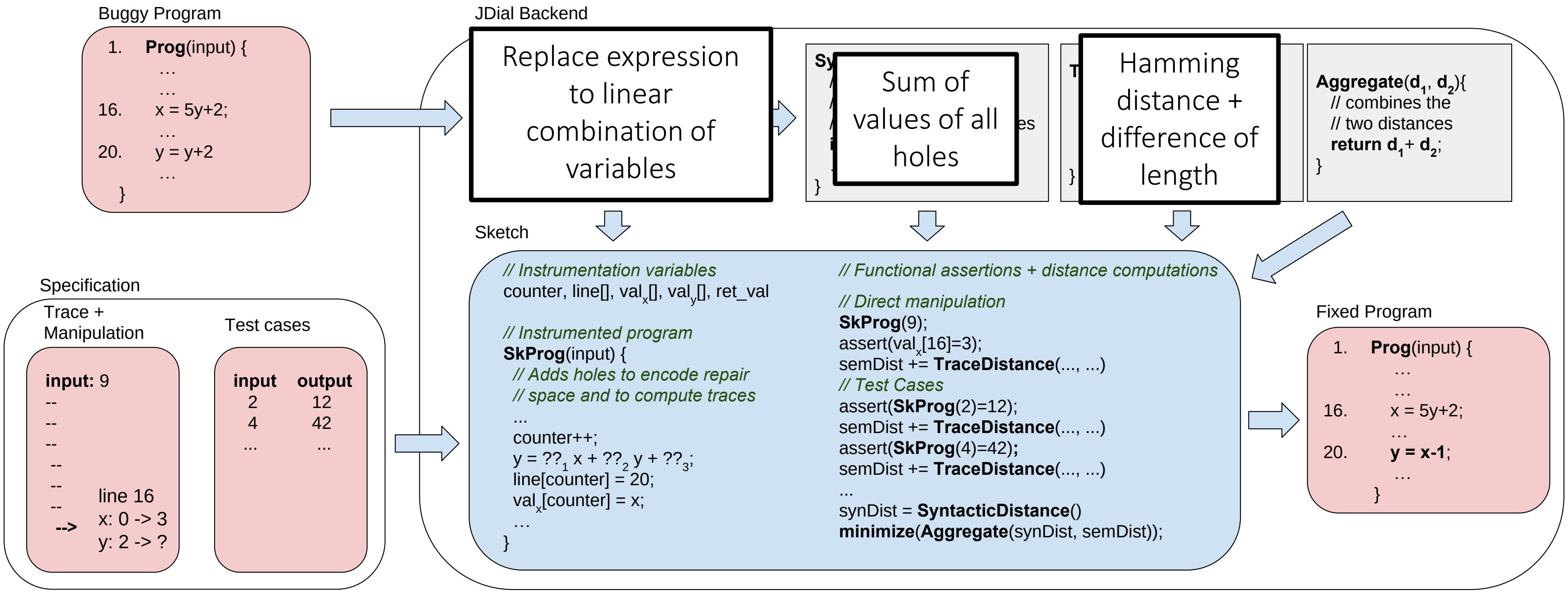
// Instrumented program
SkProg(input) {
// Adds holes to encode repair
// space and to compute traces
…
counter++;
y = ??$_1$ x + ??$_2$ y + ??$_3$;
line[counter] = 20;
val$_x$[counter] = x;
…
}

// Functional assertions + distance computations

// Direct manipulation
SkProg(9);
assert(val$_x$[16]=3);
semDist += **TraceDistance**(..., ...)
// Test Cases
assert(**SkProg**(2)=12);
semDist += **TraceDistance**(..., ...)
assert(**SkProg**(4)=42)**;**
semDist += **TraceDistance**(..., ...)
...
synDist = **SyntacticDistance**()
minimize(**Aggregate**(synDist, semDist));

The tool JDial

```
max = input[i] + ??*max + ?? * i+ ??
```



Buggy Program

```
1.    Prog(input) {
        …
        …
16.     x = 5y+2;
        …
20.     y = y+2
        …
  }
```

JDial Backend

Replace expression to linear combination of variables

Sy / / / i } es

Sum of values of all holes

T }

Hamming distance + difference of length

```
Aggregate(d_1, d_2){
  // combines the
  // two distances
  return d_1+ d_2;
}
```

Sketch

```
// Instrumentation variables
counter, line[], val_x[], val_y[], ret_val

// Instrumented program
SkProg(input) {
  // Adds holes to encode repair
  // space and to compute traces
  …
  counter++;
  y = ??_1 x + ??_2 y + ??_3;
  line[counter] = 20;
  val_x[counter] = x;
  …
}
```

```
// Functional assertions + distance computations

// Direct manipulation
SkProg(9);
assert(val_x[16]=3);
semDist += TraceDistance(..., ...)
// Test Cases
assert(SkProg(2)=12);
semDist += TraceDistance(..., ...)
assert(SkProg(4)=42);
semDist += TraceDistance(..., ...)
...
synDist = SyntacticDistance()
minimize(Aggregate(synDist, semDist));
```

Specification

Trace + Manipulation

```
input: 9
--
--
--
 --
 --
 --      line 16
 -->     x: 0 -> 3
         y: 2 -> ?
```

Test cases

input	output
2	12
4	42
...	...

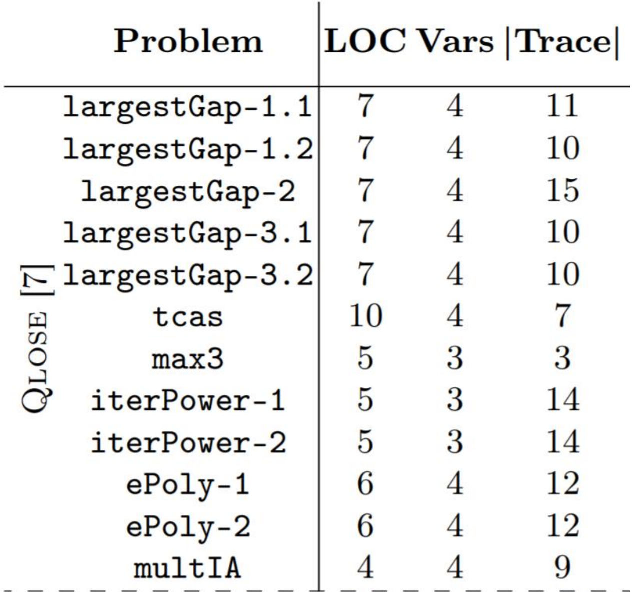
Fixed Program

```
1.    Prog(input) {
        …
        …
16.     x = 5y+2;
        …
20.     y = x-1;
        …
  }
```

Evaluation

Benchmarks

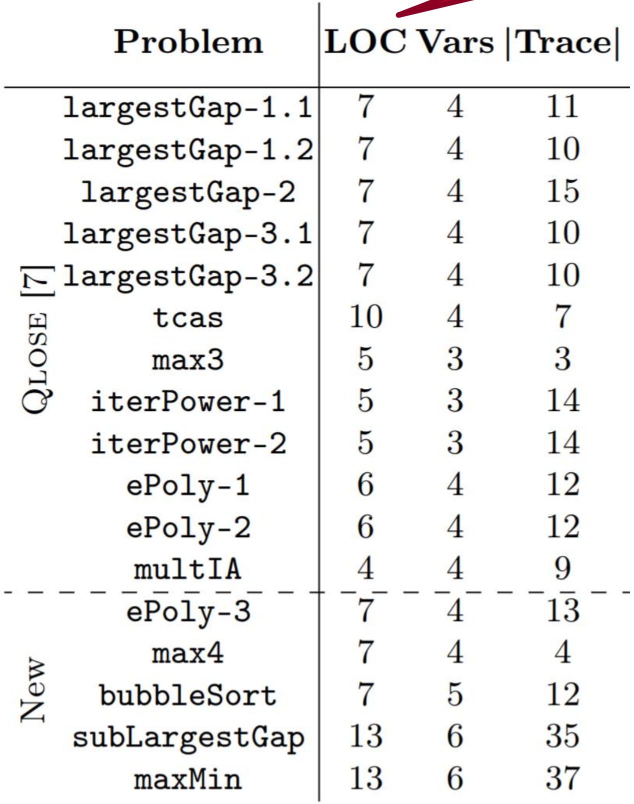
12



	Problem	LOC	Vars	|Trace|
QLOSE [7]	largestGap-1.1	7	4	11
	largestGap-1.2	7	4	10
	largestGap-2	7	4	15
	largestGap-3.1	7	4	10
	largestGap-3.2	7	4	10
	tcas	10	4	7
	max3	5	3	3
	iterPower-1	5	3	14
	iterPower-2	5	3	14
	ePoly-1	6	4	12
	ePoly-2	6	4	12
	multIA	4	4	9

Benchmarks

Avg LOC: 6.9



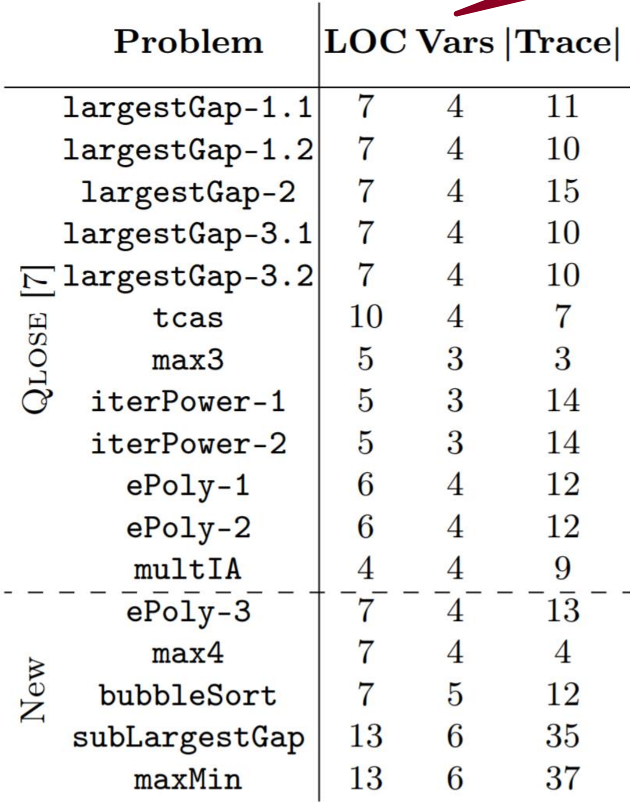
	Problem	LOC	Vars	|Trace|
QLOSE [7]	largestGap-1.1	7	4	11
	largestGap-1.2	7	4	10
	largestGap-2	7	4	15
	largestGap-3.1	7	4	10
	largestGap-3.2	7	4	10
	tcas	10	4	7
	max3	5	3	3
	iterPower-1	5	3	14
	iterPower-2	5	3	14
	ePoly-1	6	4	12
	ePoly-2	6	4	12
	multIA	4	4	9
New	ePoly-3	7	4	13
	max4	7	4	4
	bubbleSort	7	5	12
	subLargestGap	13	6	35
	maxMin	13	6	37

12

5

Benchmarks

Avg Vars: 4.1



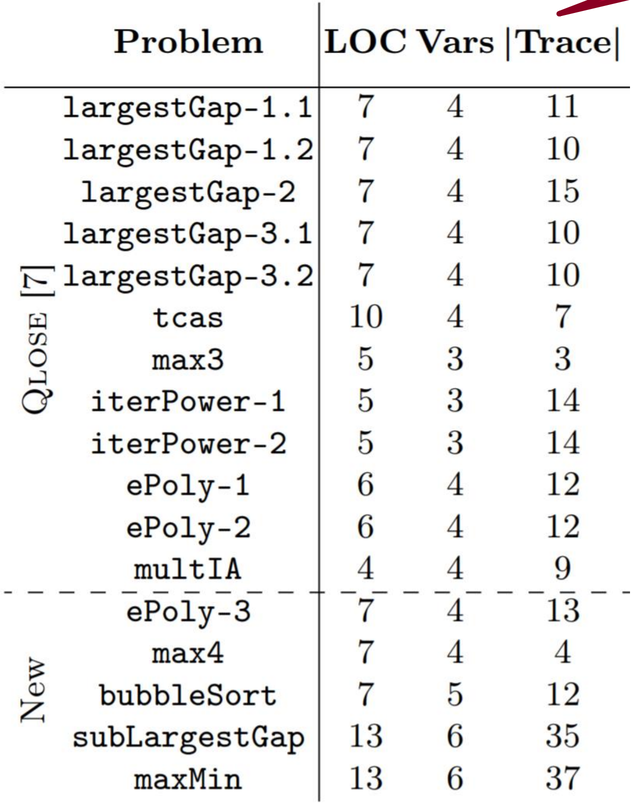
	Problem	LOC	Vars	|Trace|
QLOSE [7]	largestGap-1.1	7	4	11
	largestGap-1.2	7	4	10
	largestGap-2	7	4	15
	largestGap-3.1	7	4	10
	largestGap-3.2	7	4	10
	tcas	10	4	7
	max3	5	3	3
	iterPower-1	5	3	14
	iterPower-2	5	3	14
	ePoly-1	6	4	12
	ePoly-2	6	4	12
	multIA	4	4	9
New	ePoly-3	7	4	13
	max4	7	4	4
	bubbleSort	7	5	12
	subLargestGap	13	6	35
	maxMin	13	6	37

12

5

Benchmarks

Avg |Trace|: 17.4

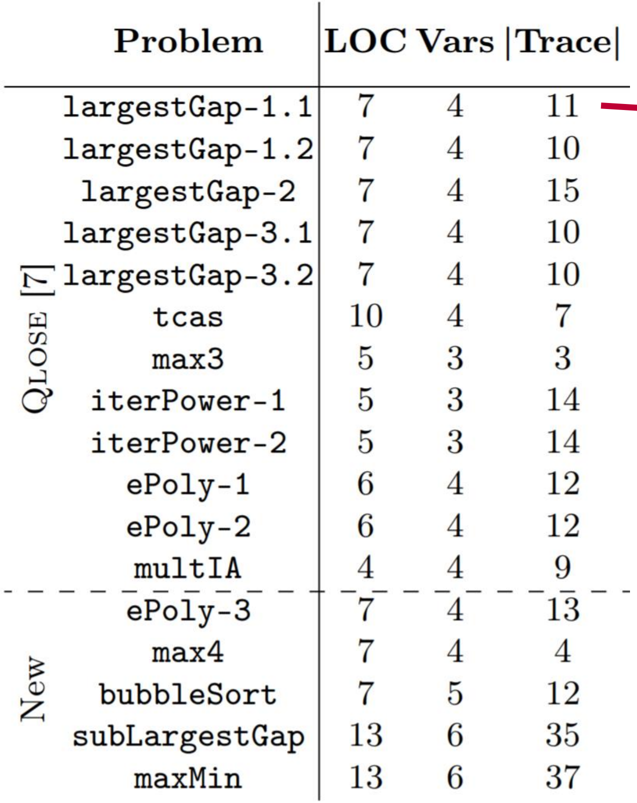


	Problem	LOC	Vars	|Trace|
Qlose [7]	largestGap-1.1	7	4	11
	largestGap-1.2	7	4	10
	largestGap-2	7	4	15
	largestGap-3.1	7	4	10
	largestGap-3.2	7	4	10
	tcas	10	4	7
	max3	5	3	3
	iterPower-1	5	3	14
	iterPower-2	5	3	14
	ePoly-1	6	4	12
	ePoly-2	6	4	12
	multIA	4	4	9
New	ePoly-3	7	4	13
	max4	7	4	4
	bubbleSort	7	5	12
	subLargestGap	13	6	35
	maxMin	13	6	37

12

5

Benchmarks

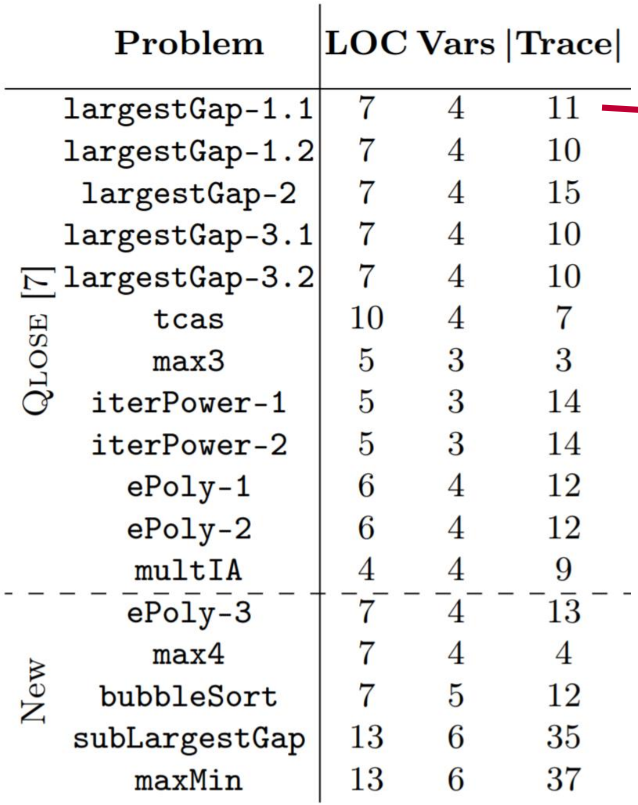


	Problem	LOC	Vars	|Trace|
QLOSE [7]	largestGap-1.1	7	4	11
	largestGap-1.2	7	4	10
	largestGap-2	7	4	15
	largestGap-3.1	7	4	10
	largestGap-3.2	7	4	10
	tcas	10	4	7
	max3	5	3	3
	iterPower-1	5	3	14
	iterPower-2	5	3	14
	ePoly-1	6	4	12
	ePoly-2	6	4	12
	multIA	4	4	9
New	ePoly-3	7	4	13
	max4	7	4	4
	bubbleSort	7	5	12
	subLargestGap	13	6	35
	maxMin	13	6	37

```
public static int lar   stGap(int[]
input){
    int max = 0;
    int min = 100;
    for(int i = 1;i < input.length;i++){
        if(input[i] > max){
         max = input[i];
        }
        if(input[i] < min){
         min = input[i];
        }

    }
    return max-min;
}
```

Incorrect initialization

Benchmarks

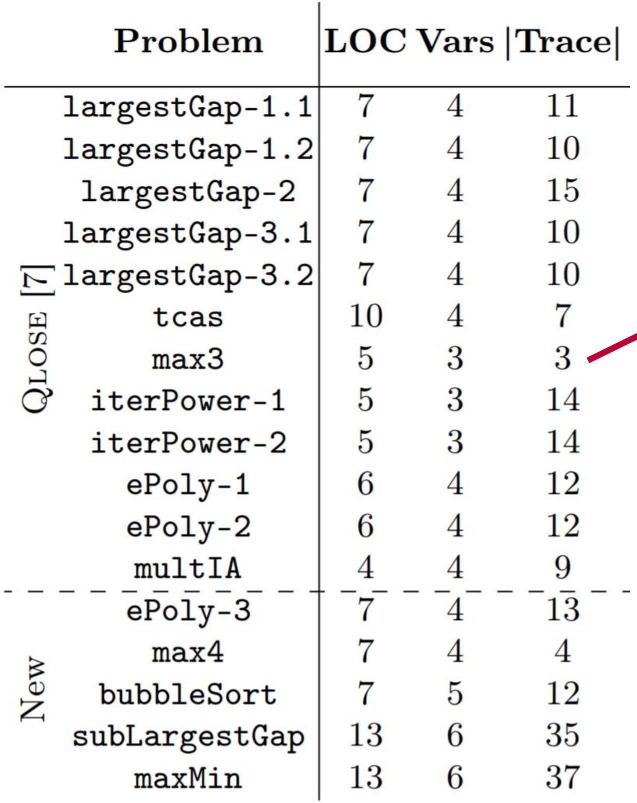


	Problem	LOC	Vars	|Trace|
QLOSE [7]	largestGap-1.1	7	4	11
	largestGap-1.2	7	4	10
	largestGap-2	7	4	15
	largestGap-3.1	7	4	10
	largestGap-3.2	7	4	10
	tcas	10	4	7
	max3	5	3	3
	iterPower-1	5	3	14
	iterPower-2	5	3	14
	ePoly-1	6	4	12
	ePoly-2	6	4	12
	multIA	4	4	9
New	ePoly-3	7	4	13
	max4	7	4	4
	bubbleSort	7	5	12
	subLargestGap	13	6	35
	maxMin	13	6	37

```
public static int largest    (int[]
input){
    int max = 0;
    int min = 100;
    for(int i = 1;i < input.length;i++){
        if(input[i] > max){
         max = input[i];
        }
        if(input[i] < min){
         min = input[i];
        }

    }
    return max-min;
}
```

Incorrect loop condition

Benchmarks

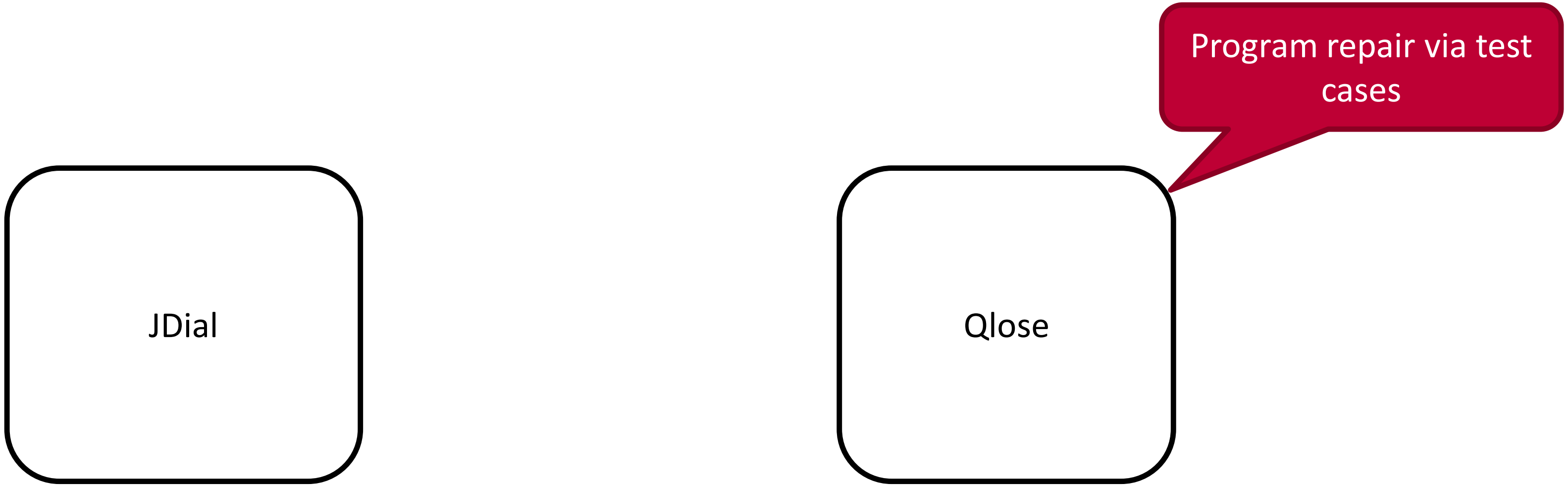


	Problem	LOC	Vars	|Trace|
QLOSE [7]	largestGap-1.1	7	4	11
	largestGap-1.2	7	4	10
	largestGap-2	7	4	15
	largestGap-3.1	7	4	10
	largestGap-3.2	7	4	10
	tcas	10	4	7
	max3	5	3	3
	iterPower-1	5	3	14
	iterPower-2	5	3	14
	ePoly-1	6	4	12
	ePoly-2	6	4	12
	multIA	4	4	9
New	ePoly-3	7	4	13
	max4	7	4	4
	bubbleSort	7	5	12
	subLargestGap	13	6	35
	maxMin	13	6	37

```
public static int ma                   z){
          if(x > y){
           y = x;
          }
          if(y > z){
           z = x;
          }
          return z;
}
```

Incorrect assignment

Question 1: Can JDial produce better repairs than Qlose*?



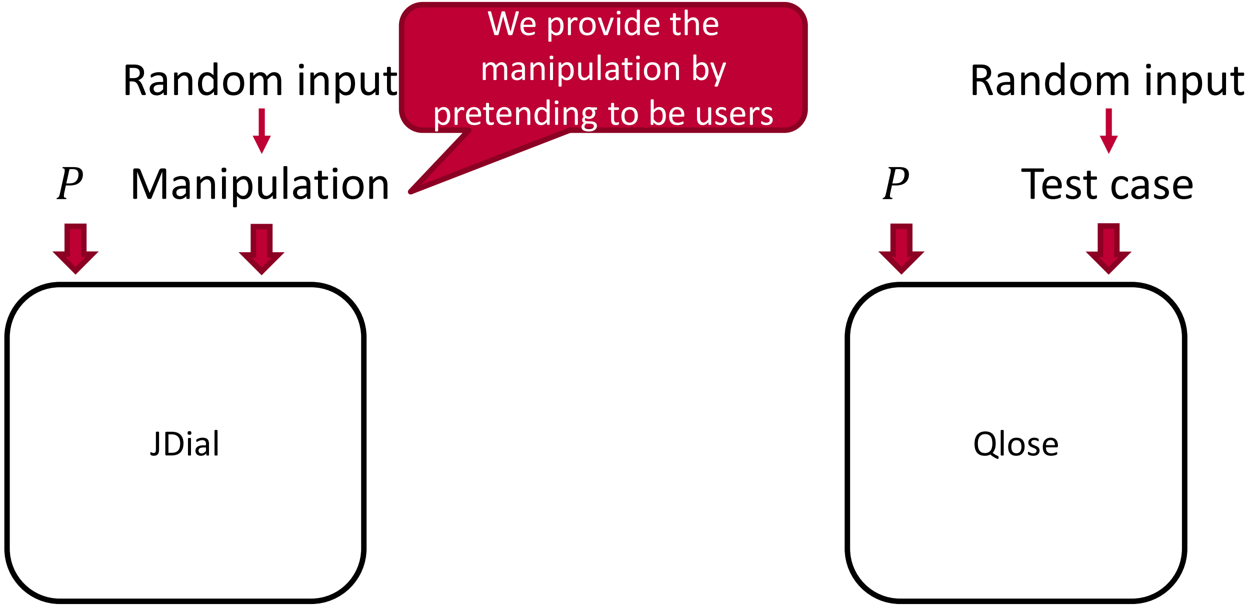
JDial

Qlose

Program repair via test cases

* D’Antoni et al, Qlose: program repair with quantitative objectives [CAV16]

Question 1: Can JDial produce better repairs than Qlose*?



Random input

P Manipulation

We provide the manipulation by pretending to be users

JDial

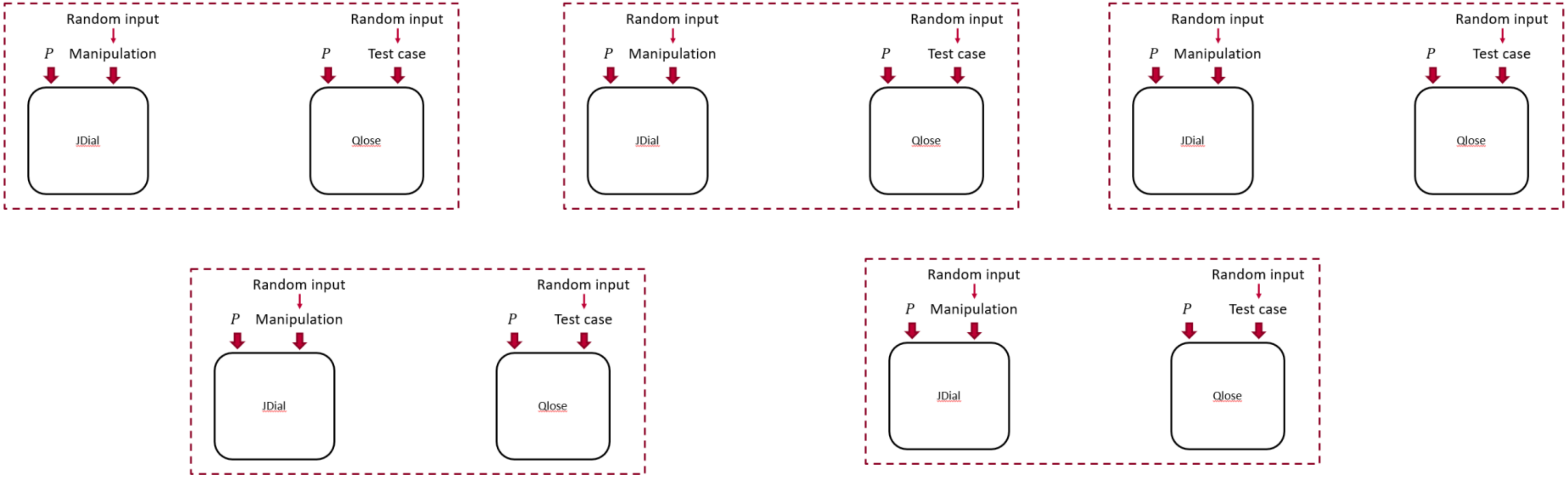
Random input

P Test case

Qlose

* D’Antoni et al, Qlose: program repair with quantitative objectives [CAV16]

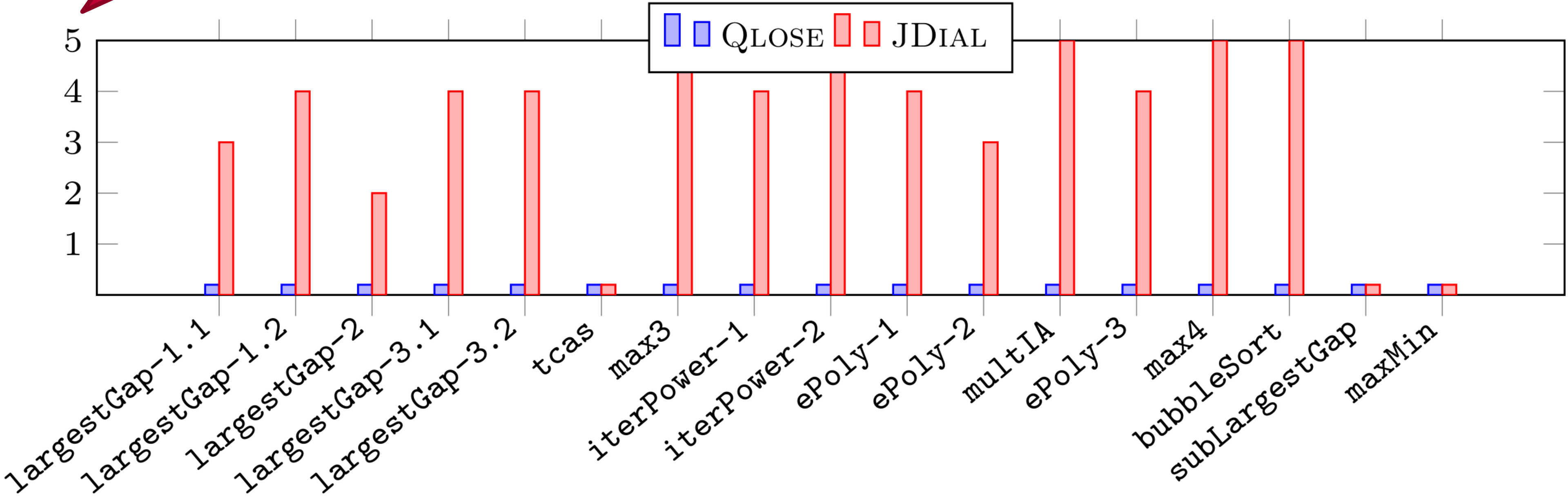
Question 1: Can JDial produce better repairs than Qlose*?



5 different random inputs

On how many random inputs each tool can solve the benchmark

test case



Avoid overfitting for single test case

Manipulated location

Repair via manipulation

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 0;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i];}
6   }
7   return max;   }
```

Trace on input = {9,5,6}

loc	1	2	3	4	5	3
i	-	-	-	1	1	1
max	-	-	0	0	0	~~5~~ → 9

Repair via test case

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 1;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i];}
6   }
7   return max + 4;   }
```

Test cases: {9,5,6} -> 9

Avoid overfitting for single test case

Repair via manipulation

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 0;i < input.length;i++){
4       if
5        m
6   }
7   retur
```

Manipulated location

Trace on input = {9,5,6}

loc	1	2	3	4	5	3
i	-	-	-	1	1	1
						~~5~~ → 9

Program repair via test cases prefer to modify return statement

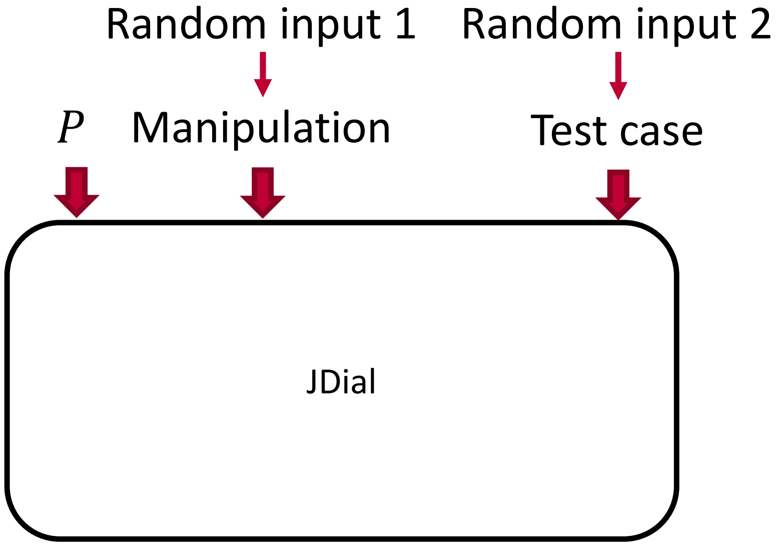
Keep all traces unchanged before return

Repair via test case

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 1;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i];}
6   }
7   return max + 4;   }
```

Test cases: {9,5,6} -> 9

Provide one more test case



Random input 1

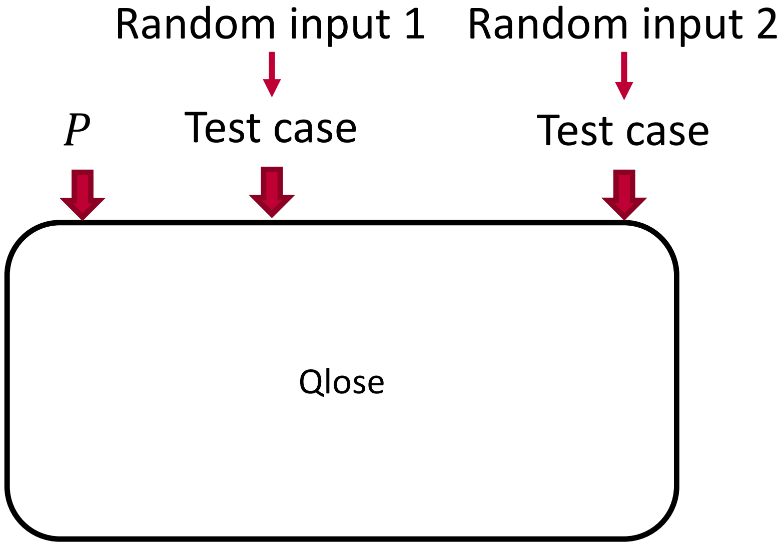
Random input 2

P

Manipulation

Test case

JDial



Random input 1

Random input 2

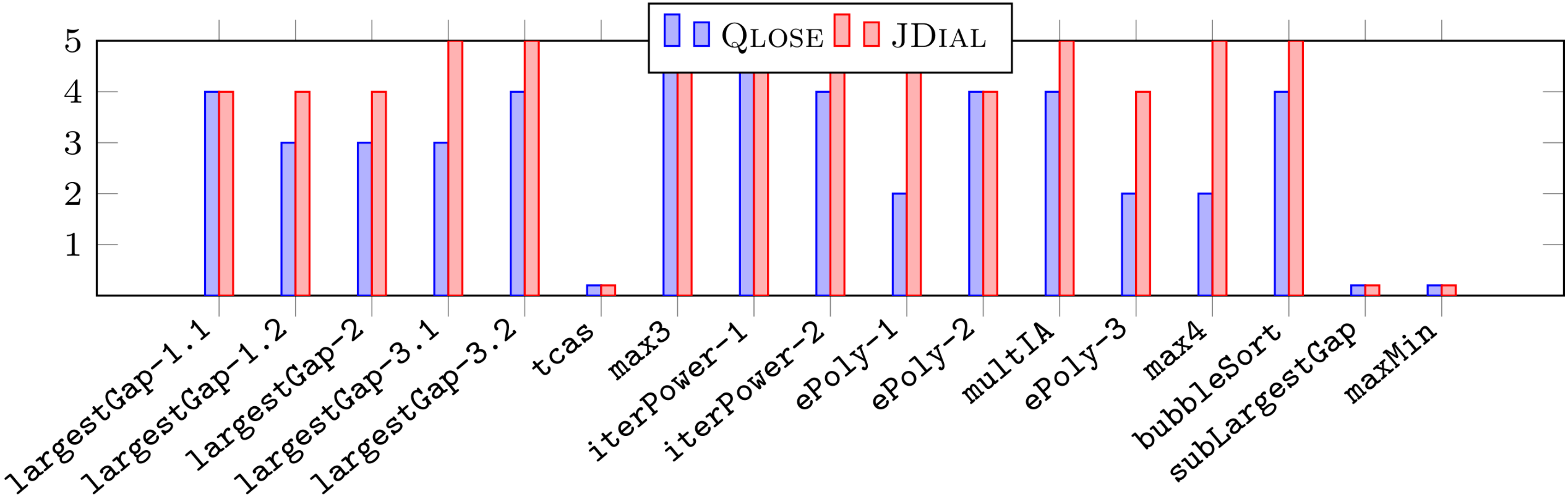
P

Test case

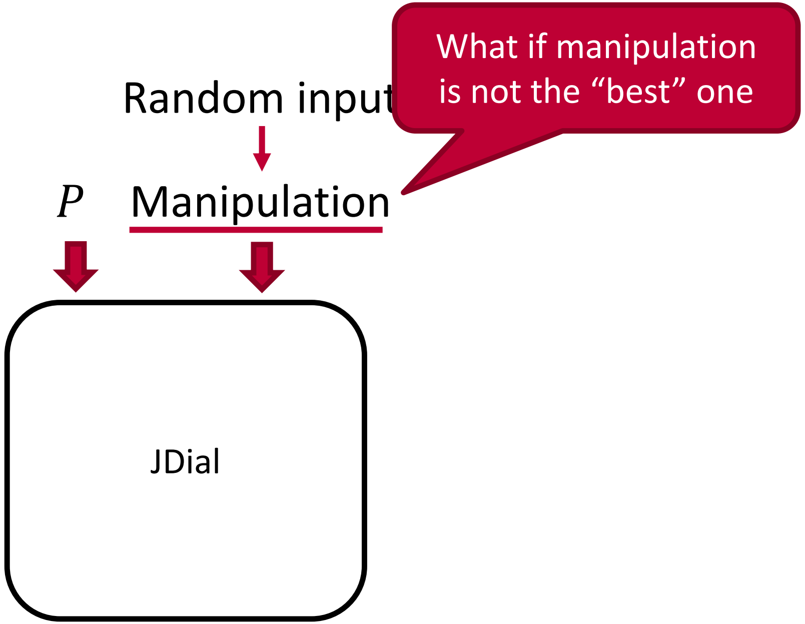
Test case

Qlose

JDial + one test case vs two test cases



Question 2: How sensitive is JDial with respect to the trace location at which the state manipulation is performed?



What if manipulation is not the "best" one

Question 2: How sensitive is JDial with respect to the trace location at which the state manipulation is performed?

```
1 public static int getMax(int[] input){
2    int max = 0;
3    for(int i = 1;i < input.length;i++){
4        if(input[i] > max){
5         max = input[i];
6        }
7    }
8    return max;
9 }
```

Trace on `input = {9,5,6,10}`

loc	1	2	3	4	5	3	← Manipulated location
i	-	-	-	1	1	1	
max	-	-	0	0	0	~~5~~	→ **9**

Question 2: How sensitive is JDial with respect to the trace location at which the state manipulation is performed?

```java
public static int getMax(int[] input){
   int max = 0;
   for(int i = 1;i < input.length;i++){
       if(input[i] > max){
        max = input[i];
       }
   }
   return max;
}
```

Trace on `input = {9,5,6,10}`

loc	1	2	3	4	← Manipulated location
i	-	-	-	~~1~~	→ **0**
max	-	-	0	0	

Question 2: How sensitive is JDial with respect to the trace location at which the state manipulation is performed?

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 1;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i];
6       }
7   }
8   return max;
9 }
```

Manipulated location

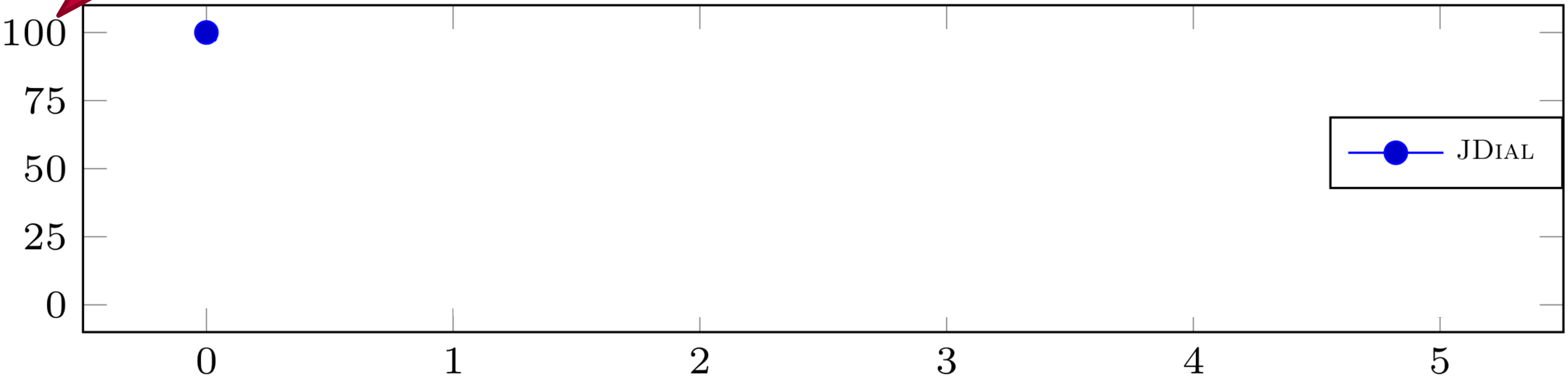
Trace on `input = {9,5,6,10}`

loc	1	2	3	4	5	3	4	5	3
i	-	-	-	1	1	1	2	2	2
max	-	-	0	0	0	5	5	5	~~6~~ → 9

We will lose the good repair with this manipulation

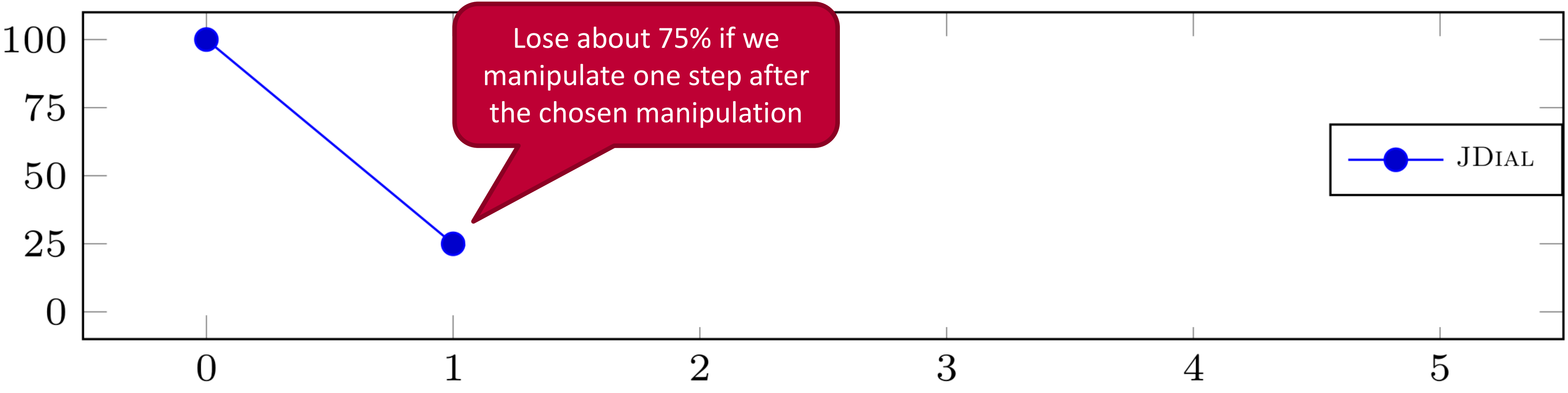
Question 2: How sensitive is JDial with respect to the trace location at anipulation is performed?

On how muchp ercentage of random inputs we lose the desired repair due to the late manipulation



Number of steps after the chosen manipulation

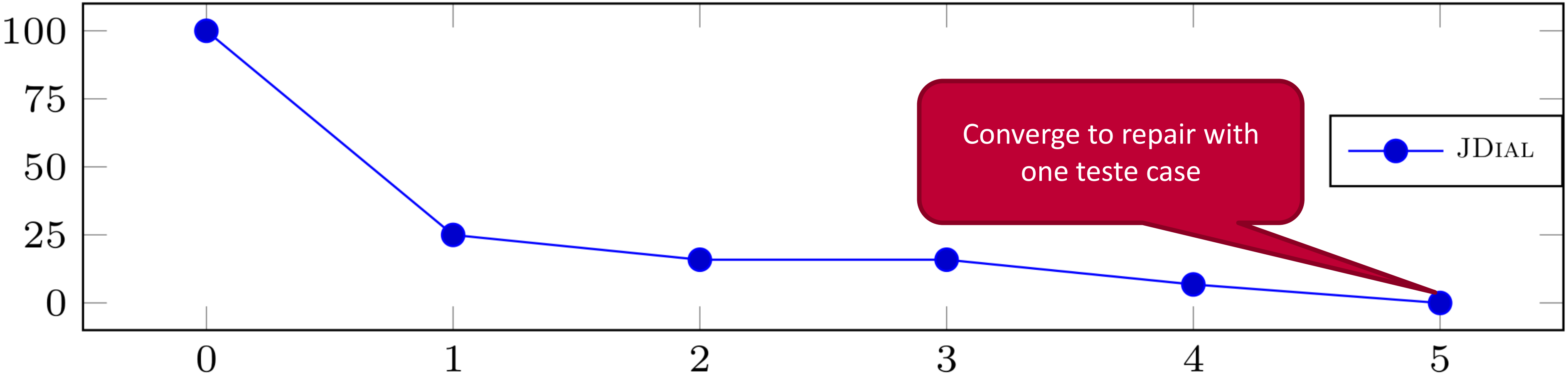
Question 2: How sensitive is JDial with respect to the trace location at which the state manipulation is performed?



Lose about 75% if we manipulate one step after the chosen manipulation

Number of steps after the chosen manipulation

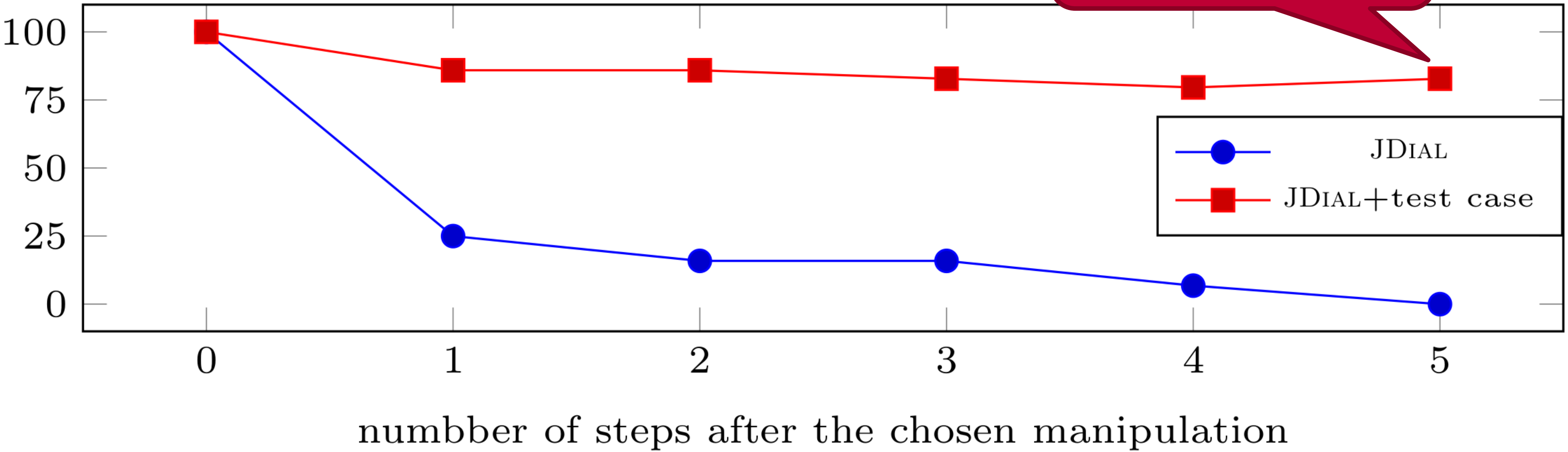
Question 2: How sensitive is JDial with respect to the trace location at which the state manipulation is performed?



Converge to repair with one teste case

Number of steps after the chosen manipulation

Question 2: How sensitive is JDial with respect to the trace location at which the state manipulation is performed?



Converge to repair with two teste cases

Comparison to program repair via test cases

Avoid overfitting for single test case

Repair via manipulation

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 0;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i];}
6   }
7   return max;   }
```

Manipulated location

Trace on input = {9,5,6}

loc	1	2	3	4	5	3
i	-	-	-	1	1	1
max	-	-	0	0	0	~~5~~ → 9

Repair via test case

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 1;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i];}
6   }
7   return max + 4;   }
```

Test cases: {9,5,6} -> 9

Avoid overfitting for single test case

Repair via manipulation

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 0;i < input.length;i++){
4       if
5        m
6   }
7   retur
```

Manipulated location

Trace on input = {9,5,6}

loc	1	2	3	4	5	3
i	-	-	-	1	1	1
						~~5~~

→ 9

Program repair via test cases prefer to modify return statement

Keep all traces unchanged before return

Repair via test case

```
1 public static int getMax(int[] input){
2   int max = 0;
3   for(int i = 1;i < input.length;i++){
4       if(input[i] > max){
5        max = input[i];}
6   }
7   return max + 4;   }
```

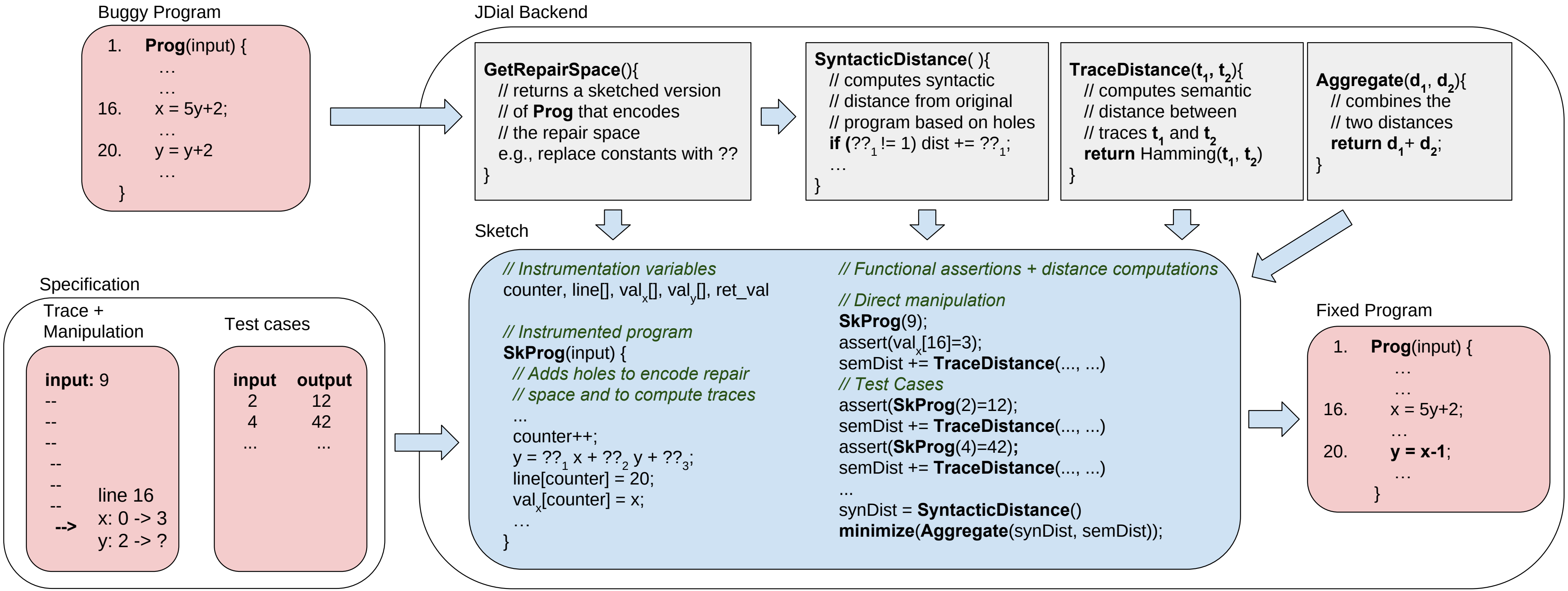
Test cases: {9,5,6} -> 9

Hard to write test cases for partial implementations

```
public static int largestGap(int[] input){
   int max = 0;
   int min = 100;
   for(int i = 1;i < input.length;i++){
      if(input[i] > max){
       max = input[i];
      }
      //TODO: implement min
   }
   int result = max-min;
   return result;
}
```

Test cases: `{9,5,6,10} -> ?`

Conclusion



Buggy Program

```
1.    Prog(input) {
          …
          …
16.      x = 5y+2;
          …
20.      y = y+2
          …
   }
```

Specification

Trace + Manipulation

```
input: 9
--
--
--
 --
 --
 --        line 16
 -->       x: 0 -> 3
           y: 2 -> ?
```

Test cases

input	output
2	12
4	42
…	…

JDial Backend

```
GetRepairSpace(){
  // returns a sketched version
  // of Prog that encodes
  // the repair space
  e.g., replace constants with ??
}
```

```
SyntacticDistance( ){
  // computes syntactic
  // distance from original
  // program based on holes
  if (??_1 != 1) dist += ??_1;
  …
}
```

```
TraceDistance(t_1, t_2){
  // computes semantic
  // distance between
  // traces t_1 and t_2
  return Hamming(t_1, t_2)
}
```

```
Aggregate(d_1, d_2){
  // combines the
  // two distances
  return d_1+ d_2;
}
```

Sketch

```
// Instrumentation variables
counter, line[], val_x[], val_y[], ret_val

// Instrumented program
SkProg(input) {
  // Adds holes to encode repair
  // space and to compute traces
  …
  counter++;
  y = ??_1 x + ??_2 y + ??_3;
  line[counter] = 20;
  val_x[counter] = x;
  …
}
```

```
// Functional assertions + distance computations

// Direct manipulation
SkProg(9);
assert(val_x[16]=3);
semDist += TraceDistance(..., ...)
// Test Cases
assert(SkProg(2)=12);
semDist += TraceDistance(..., ...)
assert(SkProg(4)=42);
semDist += TraceDistance(..., ...)
…
synDist = SyntacticDistance()
minimize(Aggregate(synDist, semDist));
```

Fixed Program

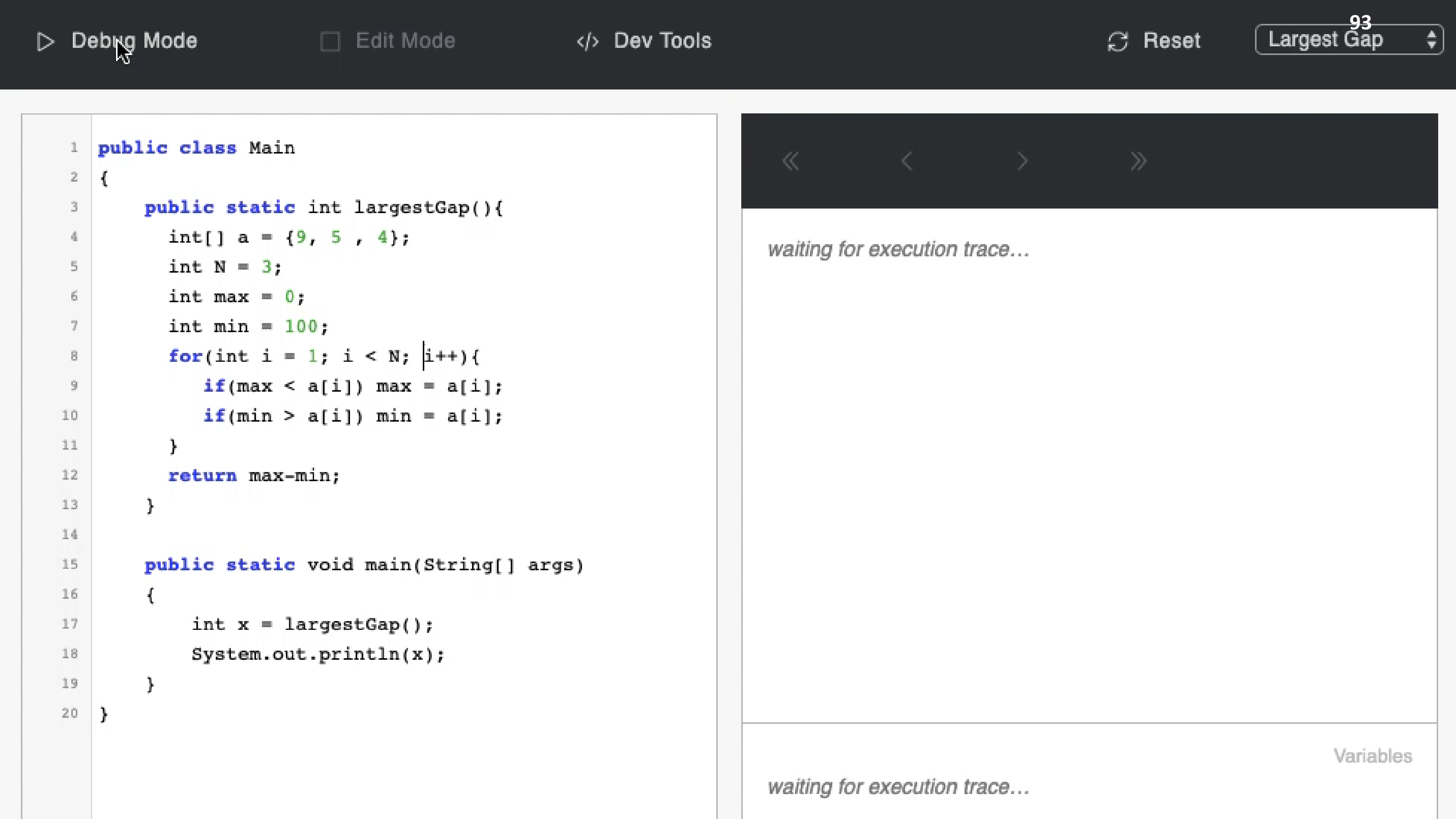
```
1.    Prog(input) {
          …
          …
16.      x = 5y+2;
          …
20.      y = x-1;
          …
   }
```

New specification mechanism that can yield better repair than test cases

JDial: a tool for repairing programs via direct state manipulation

Can we make the approach less sensitive to manipulated location?

Can JDial scale better?



```java
public class Main
{
    public static int largestGap(){
      int[] a = {9, 5 , 4};
      int N = 3;
      int max = 0;
      int min = 100;
      for(int i = 1; i < N; i++){
         if(max < a[i]) max = a[i];
         if(min > a[i]) min = a[i];
      }
      return max-min;
    }

    public static void main(String[] args)
    {
       int x = largestGap();
       System.out.println(x);
    }
}
```

waiting for execution trace…

waiting for execution trace…