加密实验

对 Proguard 和 Allatori 两种混淆工具的混淆方法进行整理，结果如表 1-1 所示。

表 1-1　Proguard 和 Allatori 混淆方法

	Proguard	Allatori
混淆方法	(1) Shrinking(删除) (2) Rename(改名) (3) Optimizer(优化) (4) Preverification(预验证)	(1) Remove(删除) (2) Renaming(改名) (3) String encryption(字符串加密) (4) Control flow obfuscation (控制流混淆) (5)Watermark(水印)

针对传统混淆技术，选取 OLLVM 和 Tigress 两种混淆工具并对它们的功能进行分类，具体如表 1-2 所示。

表 1-2　Proguard 和 Allatori 功能分类

传统混淆技术			混淆器		文献（代码混淆技术综述. 20220510）
类	子类	方法	Proguard	Allatori	
名称混淆		删除	√(1)	√(1)	[18]
		改名	√(2)	√(2)	[19-20]
控制流混淆	计算变换	不透明谓词混淆			
		冗余代码插入			
		库调用混淆			
		将可约控制流图转化为不可约控制流图			
		代码并行化			
		扩展循环条件			
		语义等价变换			
	聚合变换	方法内嵌与外联			
		方法重叠与分割			
		方法复制与代理			
		循环变换			
		分支函数			
		控制流扁平化			
	顺序变换				
数据流混淆	数据编码混淆			√(3)	[71-74]
	数据重构混淆	变量重构			
		数组重构			
		类重构			
	数据随机化				
	将静态数据转换为过程				
预防混淆	内在预防变换				
	针对性预防变换				

新型混淆技术	虚拟化			

1 Proguard

使用 Proguard 的名称混淆功能对 Java 程序进行混淆，它会将整个程序的名称变成无意义的符号。

1.1 BubbleSort

实现 BubbleSort 算法的程序为 BubbleSort.java。在 main 函数中输入待排序的一组数字，再调用 BubbleSort 方法实现冒泡排序，最后得到一组正序数字。使用"javac BubbleSort.java"得到 BubbleSort.class 文件。由于 Proguard 需要输入 jar 包，需要使用"jar cvf BubbleSort.jar BubbleSort.class"得到 BubbleSort.jar 文件。

Proguard 支持图形化的形式来实现名称混淆，具体如图 1-1 所示，得到混淆后的程序 BubbleSort_Proguard.jar（除了 main 函数名及主类名 BubbleSort 名称不变）。

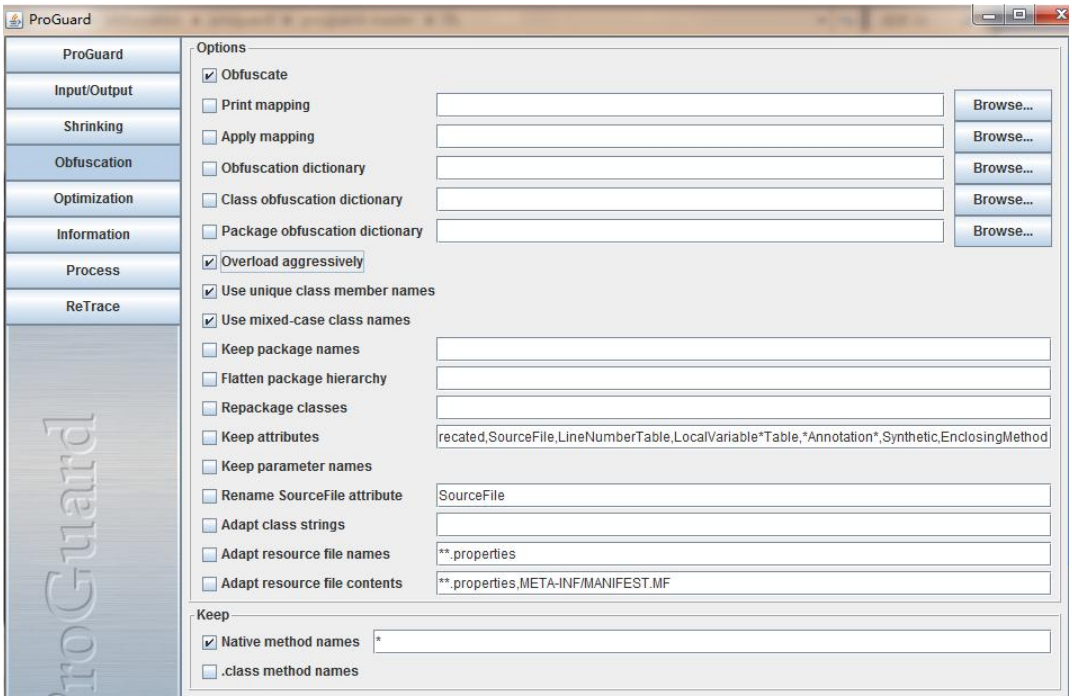

图 1-1 Proguard 混淆 BubbleSort.jar

1.1.1 正解性

分别执行原程序 BubbleSort.jar 和混淆程序 BubbleSort_Proguard.jar，以验证混淆程序的正解性。具体如图 1-2 及表 2-1 所示。

```
E:\obfuscation\1_BubbleSort\BubbleSort>java -jar BubbleSort.jar
random array:6 9 1 4 5 8 7 0 2 3
sorted array:0 1 2 3 4 5 6 7 8 9
```

```
E:\obfuscation\1_BubbleSort\BubbleSort_Proguard>java -jar BubbleSort_Proguard.ja
r
random array:6 9 1 4 5 8 7 0 2 3
sorted array:0 1 2 3 4 5 6 7 8 9
```

图 1-2 BubbleSort.jar 及 BubbleSort_Proguard.jar 运行结果

表 2-1 BubbleSort.jar 及 BubbleSort_Proguard.jar 的正解性

	BubbleSort.jar	BubbleSort_Proguard.jar
正解性	√	√

从图 1-2 可以看出，Proguard 混淆后的能正确进行排序。

1.1.2 强度

（1）程序长度

使用 SandMark 测试 BubbleSort.jar 与 BubbleSort_Proguard.jar 的程序长度，相关选项如图所示。

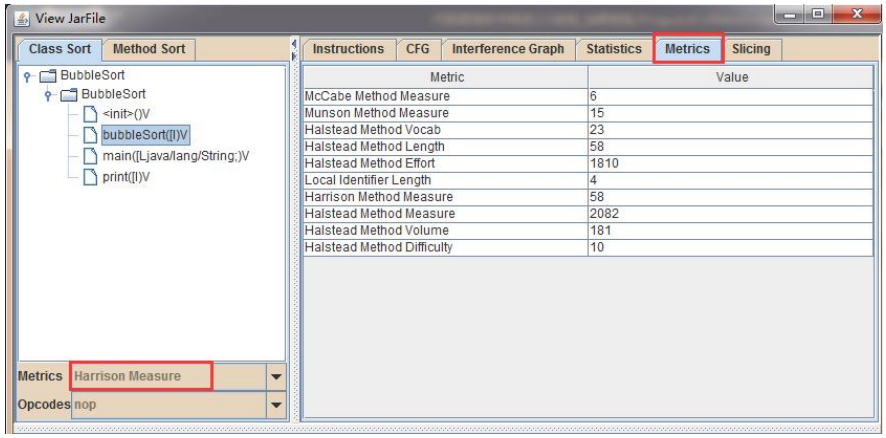

图 BubbleSort.jar 的程序长度

其中，Metrics 表示选择哪些指标。结果如表 2-2 所示。

表 2-2 BubbleSort.jar 与 BubbleSort_ Proguard.jar 程序长度

BubbleSort.jar 函数名	BubbleSort_Proguard.jar 函数名	BubbleSort.jar 程序长度	BubbleSort_Proguard.jar 程序长度
main	main	22	22
bubbleSort	c	58	58
print	d	33	33
总计		113	113

（2）控制流循环复杂度

使用 SandMark 测试 BubbleSort.jar 与 BubbleSort_Proguard.jar 的控制流循环复杂度，具体如表 2-3 所示。

表 2-3 BubbleSort.jar 与 BubbleSort_Proguard.jar 控制流循环复杂度

BubbleSort.jar 函数名	BubbleSort_Proguard.jar 函数名	BubbleSort.jar 控制流循环复杂度	BubbleSort_Proguard.jar 控制流循环复杂度
main	main	0	0
bubbleSort	c	6	6
print	d	2	2
总计		8	8

使用 Soot 测试 BubbleSort.jar 与 BubbleSort_Proguard.jar 的控制流循环复杂度，具体如表所示。

表 BubbleSort.jar 与 BubbleSort_Proguard.jar 控制流循环复杂度

BubbleSort.jar 函数名	BubbleSort_Proguard.jar 函数名	BubbleSort.jar 控制流循环复杂度	BubbleSort_Proguard.jar 控制流循环复杂度
main	main	1	1
bubbleSort	c	5	5
print	d	2	2
总计		8	8

（3）嵌套复杂度

使用 SandMark 测试 BubbleSort.jar 与 BubbleSort_Proguard.jar 的嵌套复杂度，具体如表 2-4 所示。

表 2-4 BubbleSort.jar 与 BubbleSort_Proguard.jar 嵌套复杂度

BubbleSort.jar 函数名	BubbleSort_Proguard.jar 函数名	BubbleSort.jar 嵌套复杂度	BubbleSort_Proguard.jar 嵌套复杂

			度
main	main	22	22
bubbleSort	c	58	58
print	d	33	33
总计		113	113

（4）数据结构复杂度

使用 SandMark 测试 BubbleSort.jar 与 BubbleSort_Proguard.jar 的嵌套复杂度，具体如表 2-5 所示。

表 2-5 BubbleSort.jar 与 BubbleSort_Proguard.jar 数据结构复杂度

BubbleSort.jar 函数名	BubbleSort_Proguard.jar 函数名	BubbleSort.jar 数据结构复杂度	BubbleSort_Proguard.jar 数据结构复杂度
main	main	15	15
bubbleSort	c	7	7
print	d	13	13
总计		35	35

1.1.3 执行代价

（1）程序大小

表 2-6 BubbleSort.jar 与 BubbleSort_Proguard.jar 程序大小

	BubbleSort.jar	BubbleSort_Proguard.jar
程序大小（字节）	1201	942

（2）运行时间

表 2-7 BubbleSort.jar 与 BubbleSort_Proguard.jar 运行时间

运行时间 ms	run1	run2	run3	run4	run5	run6	run7	run8	run9	run10	avg
BubbleSort.jar	16	31	15	15	16	31	31	31	32	31	24.9
BubbleSort_Proguard.jar	16	16	32	31	31	16	47	15	16	31	25.1

1.1.4 隐蔽性

使用 SandMark 测试 BubbleSort.jar 与 BubbleSort_Proguard.jar 的相似性。相似性的相关选项如下图所示。

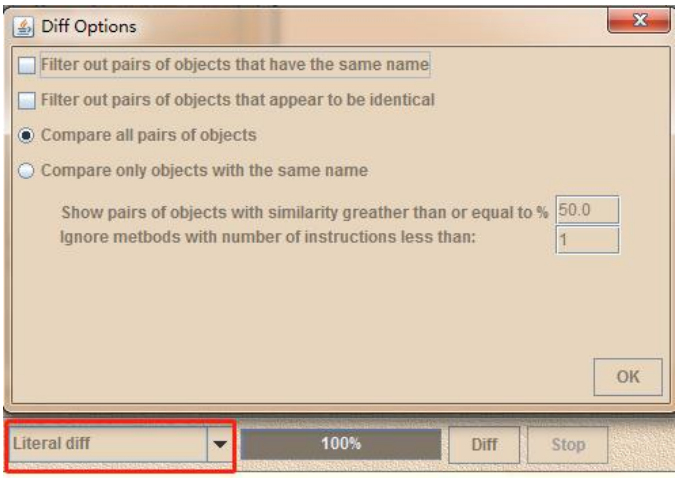

图 SandMark 相似性选项

其中，“Filter out pairs of objects that have the same name”表示去除相同名称的程序对象（如方法，类），“Filter out pairs of objects that appear to be identical”表示去除完全一样的程序对象（如方法，类），“Compare all

pairs of objects”表示比较所有的对象，“Compare only objects with the same name”表示比较相同名称的对象，“Show pairs of objects with similarity greater than or equal to filter”表示只显示相似性高于某个比例的程序对象，“Ignore methods with number of instructions less than: n.”表示忽略指令数低于某个数的函数。此处需要勾选比较所有对象，由于只改变了名称，所以只显示相似度高于 50%的函数。此外，红框部分使用“Literal diff”表示基于最长公共子序列比较不同函数的相似程度（最长公共子序列指不连续的最长公共字符串，它们位置必须相同）。比较结果如表 2-8 所示。

表 2-8 BubbleSort.jar 与 BubbleSort_ Proguard.jar 的相似性

BubbleSort.jar 函数名	BubbleSort_ Proguard.jar 函数名	BubbleSort.jar 与 BubbleSort_ Proguard.jar 相似性
main	main	100%
bubbleSort	c	100%
print	d	100%
Avg similarity		100%

1.2 QuickSort

实现 QuickSort 算法的程序为 QuickSort.java。在 main 函数中输入待排序的一组数字，再调用 QuickSort 方法实现快速排序，最后得到一组正序数字。使用”javac QuickSort.java”得到 QuickSort.class 文件。由于 Proguard 需要输入 jar 包，需要使用“jar cvf QuickSort.jar QuickSort.class”得到 QuickSort.jar 文件。

Proguard 支持图形化的形式来实现名称混淆，具体如图 2-1 所示，得到混淆后的程序 QuickSort_Proguard.jar（除了 main 函数名及主类名 QuickSort 名称不变）。

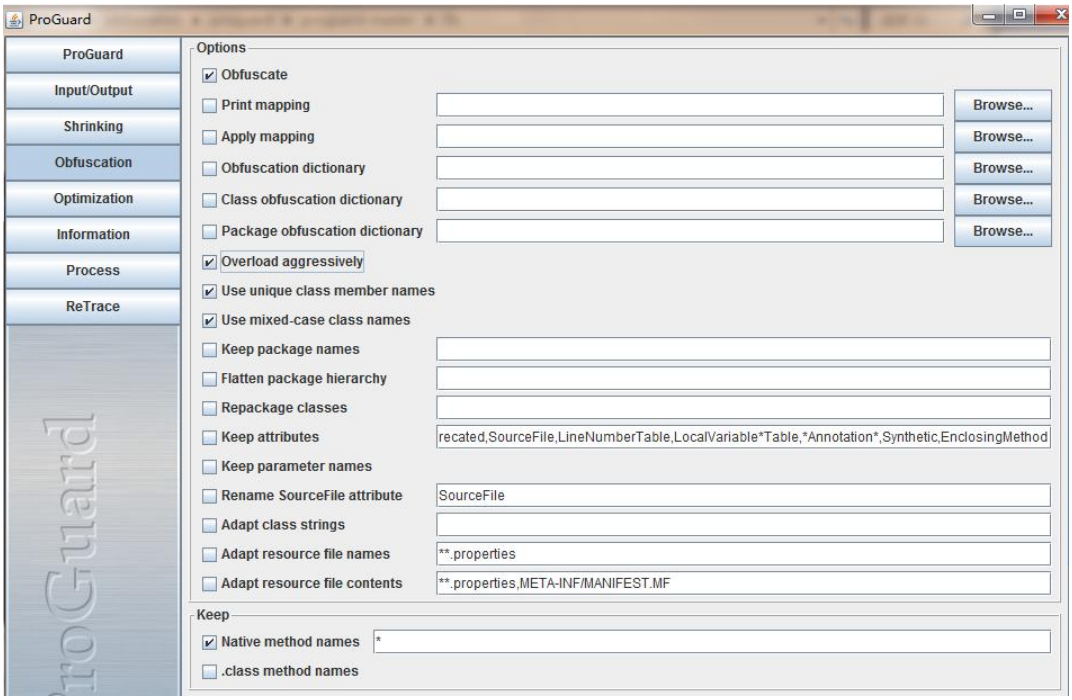

图 2-1 Proguard 混淆 QuickSort.jar

1.2.1 正解性

分别执行原程序 QuickSort.jar 和混淆程序 QuickSort_Proguard.jar，以验证混淆程序的正解性。具体如图 2-2 及表 3-1 所示。

```
E:\obfuscation\2_QuickSort\QuickSort>java -jar QuickSort.jar
random array:72 6 57 88 60 42 83 73 48 85
sorted array:6 42 48 57 60 72 73 83 85 88
```

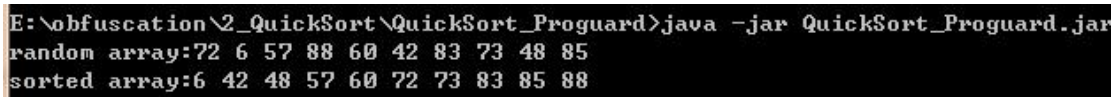

图 2-2 QuickSort.jar 及 QuickSort_Proguard.jar 运行结果

表 3-1 QuickSort.jar 及 QuickSort_Proguard.jar 的正解性

	QuickSort.jar	QuickSort_Proguard.jar
正解性	√	√

从图 2-2 可以看出，Proguard 混淆后的能正确进行排序。

1.2.2 强度

（1）程序长度

使用 SandMark 测试 QuickSort.jar 及 QuickSort_Proguard.jar 的程序长度，结果如表 3-2 所示。

表 3-2 QuickSort.jar 及 QuickSort_Proguard.jar 程序长度

QuickSort.jar 函数名	QuickSort_Proguard.jar 函数名	QuickSort.jar 程序长度	QuickSort_Proguard.jar 程序长度
quicksort	a	87	87
print	d	33	33
main	main	24	24
总计		144	144

（2）控制流循环复杂度

使用 SandMark 测试 QuickSort.jar 及 QuickSort_Proguard.jar 的控制流循环复杂度，结果如表 3-3 所示。

表 3-3 QuickSort.jar 及 QuickSort_Proguard.jar 控制流循环复杂度

QuickSort.jar 函数名	QuickSort_Proguard.jar 函数名	QuickSort.jar 控制流循环复杂度	QuickSort_Proguard.jar 控制流循环复杂度
quicksort	a	8	8
print	d	2	2
main	main	0	0
总计		10	10

使用 Soot 测试 QuickSort.jar 及 QuickSort_Proguard.jar 的控制流循环复杂度，结果如表所示。

表　QuickSort.jar 及 QuickSort_Proguard.jar 控制流循环复杂度

QuickSort.jar 函数名	QuickSort_Proguard.jar 函数名	QuickSort.jar 控制流循环复杂度	QuickSort_Proguard.jar 控制流循环复杂度
quicksort	a	8	8
print	d	2	2
main	main	1	1
总计		11	11

（3）嵌套复杂度

使用 SandMark 测试 QuickSort.jar 及 QuickSort_Proguard.jar 的嵌套复杂度，具体如表 3-4 所示。

表 3-4 QuickSort.jar 及 QuickSort_Proguard.jar 嵌套复杂度

QuickSort.jar 函数名	QuickSort_Proguard.jar 函数名	QuickSort.jar 嵌套复杂度	QuickSort_Proguard.jar 嵌套复杂度
quicksort	a	87	87
print	d	33	33

main	main	24	24
总计		144	144

（4）数据结构复杂度

使用 SandMark 测试 QuickSort.jar 与 QuickSort_Proguard.jar 的嵌套复杂度，具体如表 3-5 所示。

表 3-5 QuickSort.jar 与 QuickSort_Proguard.jar 数据结构复杂度

QuickSort.jar 函数名	QuickSort_Proguard.jar 函数名	QuickSort.jar 数据结构复杂度	QuickSort_Proguard.jar 数据结构复杂度
quicksort	a	17	17
print	d	13	13
main	main	7	7
总计		37	37

1.2.3 执行代价

（1）程序大小

表 3-6 QuickSort.jar 与 QuickSort_Proguard.jar 程序大小

	QuickSort.jar	QuickSort_Proguard.jar
程序大小（字节）	1255	967

（2）运行时间

表 3-7 QuickSort.jar 与 QuickSort_Proguard.jar 运行时间

运行时间 ms	run1	run2	run3	run4	run5	run6	run7	run8	run9	run10	avg
QuickSort.jar	31	15	15	31	32	16	31	31	31	32	26.5
QuickSort_Proguard.jar	15	31	32	16	31	31	15	32	15	32	25

1.2.4 隐蔽性

使用 SandMark 测试 QuickSort.jar 与 QuickSort_Proguard.jar 的相似性。比较结果如表 3-8 所示。

表 3-8 QuickSort.jar 与 QuickSort_ Proguard.jar 的相似性

QuickSort.jar 函数名	QuickSort_ Proguard.jar 函数名	QuickSort.jar 与 QuickSort_ Proguard.jar 相似性
quicksort	a	100%
print	d	100%
main	main	100%
Avg similarity		100%

1.3 BinarySort

实现 BinarySort 算法的程序为 BinarySort.java。在 main 函数中输入待排序的一组数字，再调用 BinarySort 方法实现二分排序，最后得到一组正序数字。使用”javac BinarySort.java”得到 BinarySort.class 文件。由于 Proguard 需要输入 jar 包，需要使用“jar cvf BinarySort.jar BinarySort.class”得到 BinarySort.jar 文件。

Proguard 支持图形化的形式来实现名称混淆，具体如图 3-1 所示，得到混淆后的程序 BinarySort_Proguard.jar（除了 main 函数名及主类名 BinarySort 名称不变）。

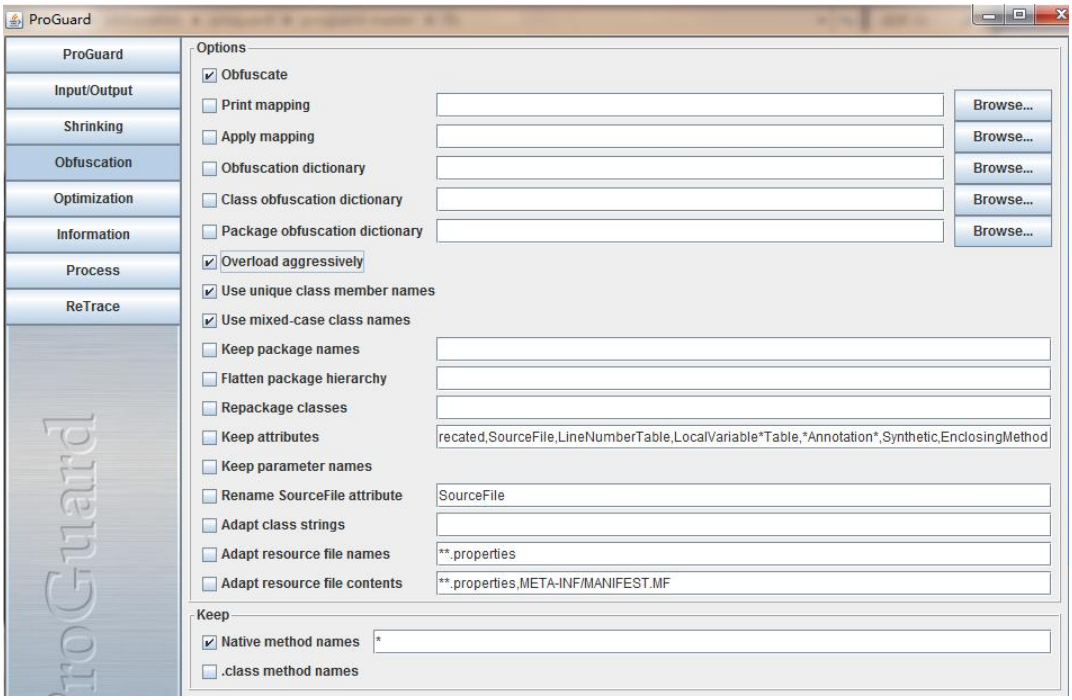

图 3-1 Proguard 混淆 BinarySort.jar

1.3.1 正解性

分别执行原程序 BinarySort.jar 和混淆程序 BinarySort_Proguard.jar，以验证混淆程序的正解性。具体如图 3-2 及表 4-1 所示。

```
E:\obfuscation\3_BinarySort\BinarySort>java -jar BinarySort.jar
random array:   12      15      9       14      4       18      23      6
sorted array:   4       6       9       12      14      15      18      23

E:\obfuscation\3_BinarySort\BinarySort_Proguard>java -jar BinarySort_Proguard.ja
r
random array:   12      15      9       14      4       18      23      6
sorted array:   4       6       9       12      14      15      18      23
```

图 3-2 BinarySort.jar 及 BinarySort_Proguard.jar 运行结果

表 4-1 BinarySort.jar 及 BinarySort_Proguard.jar 的正解性

	BinarySort.jar	BinarySort_Proguard.jar
正解性	√	√

从图 3-2 可以看出，Proguard 混淆后的能正确进行排序。

1.3.2 强度

（1）程序长度

使用 SandMark 测试 BinarySort.jar 及 BinarySort_Proguard.jar 的程序长度，结果如表 4-2 所示。

表 4-2 BinarySort.jar 及 BinarySort_Proguard.jar 程序长度

BinarySort.jar 函数名	BinarySort_Proguard.jar 函数名	BinarySort.jar 程序长度	BinarySort_Proguard.jar 程序长度
binarySort	a	70	70
printArray	b	21	21
main	main	20	20
总计		111	111

（2）控制流循环复杂度

使用 SandMark 测试 BinarySort.jar 及 BinarySort_Proguard.jar 的控制流循环复杂度，结果如表 4-3 所示。

表 4-3 BinarySort.jar 及 BinarySort_Proguard.jar 控制流循环复杂度

BinarySort.jar 函数名	BinarySort_Proguard.jar 函数名	BinarySort.jar 控制流循环复杂度	BinarySort_Proguard.jar 控制流循环复杂度
BinarySort	a	4	4
printArray	b	1	1
main	main	0	0
总计		5	5

用 Soot 测试 BinarySort.jar 及 BinarySort_Proguard.jar 的控制流循环复杂度，结果如表所示。

表 BinarySort.jar 及 BinarySort_Proguard.jar 控制流循环复杂度

BinarySort.jar 函数名	BinarySort_Proguard.jar 函数名	BinarySort.jar 控制流循环复杂度	BinarySort_Proguard.jar 控制流循环复杂度
BinarySort	a	5	5
printArray	b	2	2
main	main	1	1
总计		8	8

（3）嵌套复杂度

使用 SandMark 测试 BinarySort.jar 及 BinarySort_Allatori.jar 的嵌套复杂度，具体如表 4-4 所示。

表 4-4 BinarySort.jar 及 BinarySort_Allatori.jar 嵌套复杂度

BinarySort.jar 函数名	BinarySort_Allatori.jar 函数名	BinarySort.jar 嵌套复杂度	BinarySort_Allatori.jar 嵌套复杂度
binarySort	a	70	70
printArray	b	21	21
main	main	20	20
总计		111	111

（4）数据结构复杂度

使用 SandMark 测试 QuickSort.jar 与 QuickSort_Allatori.jar 的嵌套复杂度，具体如表 4-5 所示。

表 4-5 QuickSort.jar 与 QuickSort_Allatori.jar 数据结构复杂度

BinarySort.jar 函数名	BinarySort_Allatori.jar 函数名	QuickSort.jar 数据结构复杂度	QuickSort_Allatori.jar 数据结构复杂度
binarySort	a	19	19
printArray	b	9	9
main	main	7	7
总计		35	35

1.3.3 执行代价

（1）程序大小

表 4-6 BinarySort.jar 与 BinarySort_Proguard.jar 程序大小

	BinarySort.jar	BinarySort_Proguard.jar
程序大小（字节）	1228	950

（2）运行时间

表 4-7 BinarySort.jar 与 BinarySort_Proguard.jar 运行时间

运行时间 ms	run1	run2	run3	run4	run5	run6	run7	run8	run9	run10	avg
BinarySort.jar	31	31	16	16	32	15	16	31	31	31	25
BinarySort_Proguard.jar	15	16	31	15	16	32	31	16	31	32	23.5

1.3.4 隐蔽性

使用 SandMark 测试 BinarySort.jar 与 BinarySort _Proguard.jar 的相似性。比较结果如表 4-8 所示。

表 4-8 BinarySort.jar 与 BinarySort _Proguard.jar 的相似性

BinarySort.jar 函数名	BinarySort _ Proguard.jar 函数名	BinarySort.jar 与 BinarySort _ Proguard.jar 相似性
binarySort	a	100%
printArray	b	100%
main	main	100%
Avg similarity		100%

2 Allatori

使用 Allatori 的名称混淆功能对 Java 程序混淆，它会将整个程序的名称变成无意义的符号。

2.1 BubbleSort

实现 BubbleSort 算法的程序为 BubbleSort.java。在 main 函数中输入待排序的一组数字，再调用 BubbleSort 方法实现冒泡排序，最后得到一组正序数字。使用”javac BubbleSort.java”得到 BubbleSort.class 文件。由于 Allatori 需要输入 jar 包，需要使用“jar cvf BubbleSort.jar BubbleSort.class”得到 BubbleSort.jar 文件。

Allatori 使用 config.xml 来实现名称混淆，具体如图 4-1 所示，运行 RunAllatori.bat 得到混淆后的程序 BubbleSort_Allatori.jar（除了 main 函数名及主类名 BubbleSort 名称不变）。

```
<config>
    <input>
        <jar in="BubbleSort.jar" out="BubbleSort_Allatori.jar"/>
    </input>

    <keep-names>
        <class access="protected+">
            <field access="protected+"/>
        </class>
    </keep-names>

    <property name="log-file" value="log.xml"/>

</config>
```

```
E:\obfuscation\Allatori\Allatori-7.7-Demo\tutorial\test\1_BubbleSort>java -Xms12
8m -Xmx512m -jar ..\..\..\lib\allatori.jar config.xml
```

图 4-1 Allatori 混淆 BubbleSort.jar

此外，会输出 log.xml，它是混淆的日志文件，它会涉及混淆前后名称的映射，具体如图 4-2 所示。

```
<mapping>
 <class old="BubbleSort" new="BubbleSort">
  <method old="AllatoriDecryptString(Ljava/lang/String;)Ljava/lang/String;" new="ALLATORIxDEMO"/>
  <method old="bubbleSort([I)V" new="ALLATORIxDEMO" s="3" e="18"/>
  <method old="main([Ljava/lang/String;)V" new="main" s="21" e="27"/>
  <method old="print([I)V" new="I" s="30" e="35"/>
 </class>
```

图 4-2 Allatori 混淆 BubbleSort.jar 日志文件

2.1.1 正解性

分别执行原程序 BubbleSort.jar 和混淆程序 BubbleSort_Allatori.jar，以验证混淆程序的正解性。具体如图 4-3 及表 5-1 所示。

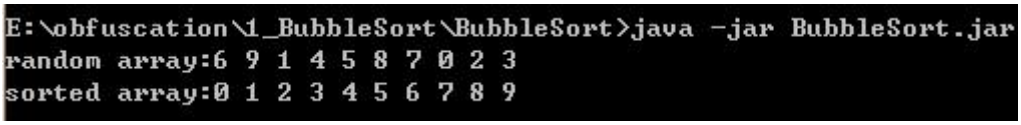

图 4-3 BubbleSort.jar 及 BubbleSort_Allatori.jar 运行结果

表 5-1 BubbleSort.jar 及 BubbleSort_Allatori.jar 的正解性

	BubbleSort.jar	BubbleSort_Allatori.jar
正解性	√	√

从图 4-3 可以看出，Allatori 混淆后的能正确进行排序。

2.1.2 强度

（1）程序长度

使用 SandMark 测试 BubbleSort.jar 与 BubbleSort_Allatori.jar 的程序长度，结果如表 5-2 所示。

表 5-2 BubbleSort.jar 与 BubbleSort_ Allatori.jar 程序长度

BubbleSort.jar 函数名	BubbleSort_Allatori.jar 函数名	BubbleSort.jar 程序长度	BubbleSort_Allatori.jar 程序长度
main	main	22	25
bubbleSort	ALLATORIxDEMO	58	56
print	m	33	33
总计		113	114

（2）控制流循环复杂度

使用 SandMark 测试 BubbleSort.jar 与 BubbleSort_Allatori.jar 的控制流循环复杂度，结果如表 5-3 所示。

表 5-3 BubbleSort.jar 与 BubbleSort_ Allatori.jar 控制流循环复杂度

BubbleSort.jar 函数名	BubbleSort_Allatori.jar 函数名	BubbleSort.jar 控制流循环复杂度	BubbleSort_Allatori.jar 控制流循环复杂度
main	main	0	0
bubbleSort	ALLATORIxDEMO	6	6
print	m	2	2
总计		8	8

使用 Soot 测试 BubbleSort.jar 与 BubbleSort_Allatori.jar 的控制流循环复杂度，结果如表所示。

表 BubbleSort.jar 与 BubbleSort_ Allatori.jar 控制流循环复杂度

BubbleSort.jar 函数名	BubbleSort_Allatori.jar 函数名	BubbleSort.jar 控制流循环复杂度	BubbleSort_Allatori.jar 控制流循环复杂度
main	main	1	1
bubbleSort	ALLATORIxDEMO	5	5
print	m	2	2
总计		8	8

（3）嵌套复杂度

使用 SandMark 测试 BubbleSort.jar 与 BubbleSort_Allatori.jar 的嵌套复杂度，具体如表 5-4 所示。

表 5-4 BubbleSort.jar 与 BubbleSort_Allatori.jar 嵌套复杂度

BubbleSort.jar 函数名	BubbleSort_Allatori.jar 函数名	BubbleSort.jar 嵌套复杂度	BubbleSort_Allatori.jar 嵌套复杂度
main	main	22	25
bubbleSort	ALLATORIxDEMO	58	56
print	m	33	33
总计		113	114

（4）数据结构复杂度

使用 SandMark 测试 BubbleSort.jar 与 BubbleSort_Allatori.jar 的嵌套复杂度，具体如表 5-5 所示。

表 5-5 BubbleSort.jar 与 BubbleSort_Allatori.jar 数据结构复杂度

BubbleSort.jar 函数名	BubbleSort_Allatori.jar 函数名	BubbleSort.jar 数据结构复杂度	BubbleSort_Allatori.jar 数据结构复杂度
main	main	15	15
bubbleSort	ALLATORIxDEMO	7	7
print	m	13	13
总计		35	35

2.1.3 执行代价

（1）程序大小

表 5-6 BubbleSort.jar 与 BubbleSort_Allatori.jar 程序大小

	BubbleSort.jar	BubbleSort_Allatori.jar（含版本信息）
程序大小（字节）	1201	1776

（2）运行时间

表 5-7 BubbleSort.jar 与 BubbleSort_Allatori.jar 运行时间

运行时间 ms	run1	run2	run3	run4	run5	run6	run7	run8	run9	run10	avg
BubbleSort.jar	16	31	15	15	16	31	31	31	32	31	24.9
BubbleSort_Allatori.jar	16	32	31	31	31	31	31	16	31	15	26.5

2.1.4 隐蔽性

使用 SandMark 测试 BubbleSort.jar 与 BubbleSort_Allatori.jar 的相似性。相似性的相关选项如下图所示。

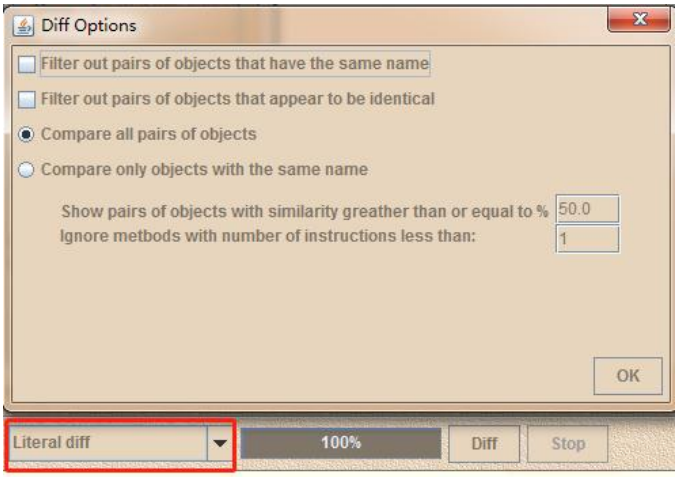

图 SandMark 相似性选项

其中，“Filter out pairs of objects that have the same name”表示去除相同名称的程序对象（如方法，类），“Filter out pairs of objects that appear to be identical”表示去除完全一样的程序对象（如方法，类），“Compare all pairs of objects”表示比较所有的对象，“Compare only objects with the same name”表示比较相同名称的对象，“Show pairs of objects with similarity greater than or equal to filter”表示只显示相似性高于某个比例的程序对象，“Ignore methods with number of instructions less than: n.”表示忽略指令数低于某个数的函数。此处需要勾选比较所有对象，由于只改变了名称，所以只显示相似度高于 50%的函数。此外，红框部分使用“Literal diff”表示基于最长公共子序列比较不同函数的相似程度（最长公共子序列指不连续的最长公共字符串，它们位置必须相同）。

比较结果如表 5-8 所示。

表 5-8 BubbleSort.jar 与 BubbleSort_Allatori.jar 的相似性

BubbleSort.jar 函数名	BubbleSort_Allatori.jar 函数名	BubbleSort.jar 与 BubbleSort_Allatori.jar 相似性
main	main	83.1%
bubbleSort	ALLATORIxDEMO	82.4%
print	m	72.2%
Avg similarity		79.23%

2.2 QuickSort

实现 QuickSort 算法的程序为 QuickSort.java。在 main 函数中输入待排序的一组数字，再调用 QuickSort 方法实现快速排序，最后得到一组正序数字。使用”javac QuickSort.java”得到 QuickSort.class 文件。由于 Allatori 需要输入 jar 包，需要使用“jar cvf QuickSort.jar QuickSort.class”得到 QuickSort.jar 文件。

Allatori 使用 config.xml 来实现名称混淆，具体如图 5-1 所示，运行 RunAllatori.bat 得到混淆后的程序 QuickSort_Allatori.jar（除了 main 函数名及主类名 QuickSort 名称不变）。

```
<config>
    <input>
        <jar in="QuickSort.jar" out="QuickSort_Allatori.jar"/>
    </input>

    <keep-names>
        <class access="protected+">
            <field access="protected+"/>
        </class>
    </keep-names>

    <property name="log-file" value="log.xml"/>
</config>
```

```
E:\obfuscation\Allatori\Allatori-7.7-Demo\tutorial\test\2_QuickSort>java -Xms128
m -Xmx512m -jar ..\..\..\lib\allatori.jar config.xml
```

图 5-1 Allatori 混淆 QuickSort.jar

此外，会输出 log.xml，它是混淆的日志文件，它会涉及混淆前后名称的映射，具体如图 5-2 所示。

```
<mapping>
 <class old="QuickSort" new="QuickSort">
  <method old="AllatoriDecryptString(Ljava/lang/String;)Ljava/lang/String;" new="ALLATORIxDEMO"/>
  <method old="main([Ljava/lang/String;)V" new="main" s="44" e="51"/>
  <method old="print([I)V" new="ALLATORIxDEMO" s="37" e="42"/>
  <method old="quickSort([III)V" new="ALLATORIxDEMO" s="4" e="34"/>
 </class>
```

图 5-2 Allatori 混淆 QuickSort.jar 日志文件

2.2.1 正解性

分别执行原程序 QuickSort.jar 和混淆程序 QuickSort_Allatori.jar，以验证混淆程序的正解性。具体如图 5-3 及表 6-1 所示。

```
E:\obfuscation\2_QuickSort\QuickSort>java -jar QuickSort.jar
random array:72 6 57 88 60 42 83 73 48 85
sorted array:6 42 48 57 60 72 73 83 85 88
```

```
E:\obfuscation\2_QuickSort\QuickSort_Allatori>java -jar QuickSort_Allatori.jar

#######################################################
#                                                     #
#         ## #   #    ## ### ### ##  ###             #
#        # # #   #   # #  #  # # # #  #               #
#        ### #   #   ###  #  # # ##   #               #
#        # # ### ### # #  #  ### # # ###              #
#                                                     #
# Obfuscation by Allatori Obfuscator v7.7 DEMO        #
#                                                     #
#            http://www.allatori.com                  #
#                                                     #
#######################################################

random array:72 6 57 88 60 42 83 73 48 85
sorted array:6 42 48 57 60 72 73 83 85 88
```

图 5-3 QuickSort.jar 及 QuickSort_Allatori.jar 运行结果

表 6-1 QuickSort.jar 及 QuickSort_Allatori.jar 的正解性

	QuickSort.jar	QuickSort_Allatori.jar
正解性	√	√

从图 5-3 可以看出，Allatori 混淆后的能正确进行排序。

2.2.2 强度

（1）程序长度

使用 SandMark 测试 QuickSort.jar 及 QuickSort_Allatori.jar 的程序长度，结果如表 6-2 所示。

表 6-2 QuickSort.jar 及 QuickSort_Allatori.jar 程序长度

QuickSort.jar 函数名	QuickSort_Allatori.jar 函数名	QuickSort.jar 程序长度	QuickSort_Allatori.jar 程序长度
quicksort	ALLATORIxDEMO	87	86
print	ALLATORIxDEMO	33	33
main	main	24	28
总计		144	147

（2）控制流循环复杂度

使用 SandMark 测试 QuickSort.jar 及 QuickSort_Allatori.jar 的控制流循环复杂度，结果如表 6-3 所示。

表 6-3 QuickSort.jar 及 QuickSort_ Allatori.jar 控制流循环复杂度

QuickSort.jar 函数名	QuickSort_ Allatori.jar 函数名	QuickSort.jar 控制流循环复	QuickSort_ Allatori.jar 控制流循

		杂度	环复杂度
quicksort	ALLATORIxDEMO	8	8
print	ALLATORIxDEMO	2	2
main	main	0	0
总计		10	10

使用 Soot 测试 QuickSort.jar 及 QuickSort_Allatori.jar 的控制流循环复杂度，结果如表所示。

表　QuickSort.jar 及 QuickSort_ Allatori.jar 控制流循环复杂度

QuickSort.jar 函数名	QuickSort_ Allatori.jar 函数名	QuickSort.jar 控制流循环复杂度	QuickSort_ Allatori.jar 控制流循环复杂度
quicksort	ALLATORIxDEMO	8	8
print	ALLATORIxDEMO	2	2
main	main	1	1
总计		11	11

（3）嵌套复杂度

使用 SandMark 测试 QuickSort.jar 及 QuickSort_Allatori.jar 的嵌套复杂度，具体如表 6-4 所示。

表 6-4 QuickSort.jar 及 QuickSort_Allatori.jar 嵌套复杂度

QuickSort.jar 函数名	QuickSort_Allatori.jar 函数名	QuickSort.jar 嵌套复杂度	QuickSort_Allatori.jar 嵌套复杂度
quicksort	ALLATORIxDEMO	87	86
print	ALLATORIxDEMO	33	33
main	main	24	28
总计		144	147

（4）数据结构复杂度

使用 SandMark 测试 QuickSort.jar 与 QuickSort_Allatori.jar 的嵌套复杂度，具体如表 6-5 所示。

表 6-5 QuickSort.jar 与 QuickSort_Allatori.jar 数据结构复杂度

QuickSort.jar 函数名	QuickSort_Allatori.jar 函数名	QuickSort.jar 数据结构复杂度	QuickSort_Allatori.jar 数据结构复杂度
quicksort	ALLATORIxDEMO	17	17
print	ALLATORIxDEMO	13	13
main	main	7	7
总计		37	37

2.2.3 执行代价

（1）程序大小

表 6-6 QuickSort.jar 与 QuickSort_Allatori.jar 程序大小

	QuickSort.jar	QuickSort_Allatori.jar（含版本信息）
程序大小（字节）	1255	1850

（2）运行时间

表 6-7 QuickSort.jar 与 QuickSort_Allatori.jar 运行时间

运行时间 ms	run1	run2	run3	run4	run5	run6	run7	run8	run9	run10	avg
QuickSort.jar	31	15	15	31	32	16	31	31	31	32	26.5
QuickSort_Allatori.jar	16	31	31	31	32	16	31	16	31	31	26.6

2.2.4 隐蔽性

使用 SandMark 测试 QuickSort.jar 与 QuickSort_Allatori.jar 的相似性。比较结果如 6-8 表所示。

表 6-8 QuickSort.jar 与 QuickSort_Allatori.jar 的相似性

QuickSort.jar 函数名	QuickSort_Allatori.jar 函数名	QuickSort.jar 与 QuickSort_Allatori.jar 相似性
quicksort	ALLATORIxDEMO	73.1%
print	ALLATORIxDEMO	72.2%
main	main	84.3%
Avg similarity		76.53%

2.3 BinarySort

实现 BinarySort 算法的程序为 BinarySort.java。在 main 函数中输入待排序的一组数字，再调用 BinarySort 方法实现二分排序，最后得到一组正序数字。使用"javac BinarySort.java"得到 BinarySort.class 文件。由于 Allatori 需要输入 jar 包，需要使用"jar cvf BinarySort.jar BinarySort.class"得到 BinarySort.jar 文件。

Allatori 使用 config.xml 来实现名称混淆，具体如图 6-1 所示，运行 RunAllatori.bat 得到混淆后的程序 BinarySort_Allatori.jar（除了 main 函数名及主类名 BinarySort 名称不变）。

```
<config>
    <input>
        <jar in="BinarySort.jar" out="BinarySort_Allatori.jar"/>
    </input>

    <keep-names>
        <class access="protected+">
            <field access="protected+"/>
        </class>
    </keep-names>

    <property name="log-file" value="log.xml"/>
</config>
```

```
E:\obfuscation\Allatori\Allatori-7.7-Demo\tutorial\test\3_BinarySort>java -Xms12
8m -Xmx512m -jar ..\..\..\lib\allatori.jar config.xml
```

图 6-1 Allatori 混淆 BinarySort.jar

此外，会输出 log.xml，它是混淆的日志文件，它会涉及混淆前后名称的映射，具体如图 6-2 所示。

```
<mapping>
 <class old="BinarySort" new="BinarySort">
  <method old="AllatoriDecryptString(Ljava/lang/String;)Ljava/lang/String;" new="ALLATORIxDEMO"/>
  <method old="binarySort([I)V" new="ALLATORIxDEMO" s="6" e="35"/>
  <method old="main([Ljava/lang/String;)V" new="main" s="45" e="53"/>
  <method old="printArray([I)V" new="j" s="38" e="42"/>
 </class>
```

图 6-2 Allatori 混淆 BinarySort.jar 日志文件

2.3.1 正解性

分别执行原程序 BinarySort.jar 和混淆程序 BinarySort_Allatori.jar，以验证混淆程序的正解性。具体如图 6-3 及表 7-1 所示。

```
E:\obfuscation\3_BinarySort\BinarySort>java -jar BinarySort.jar
random array:   12      15      9       14      4       18      23      6
sorted array:   4       6       9       12      14      15      18      23
```

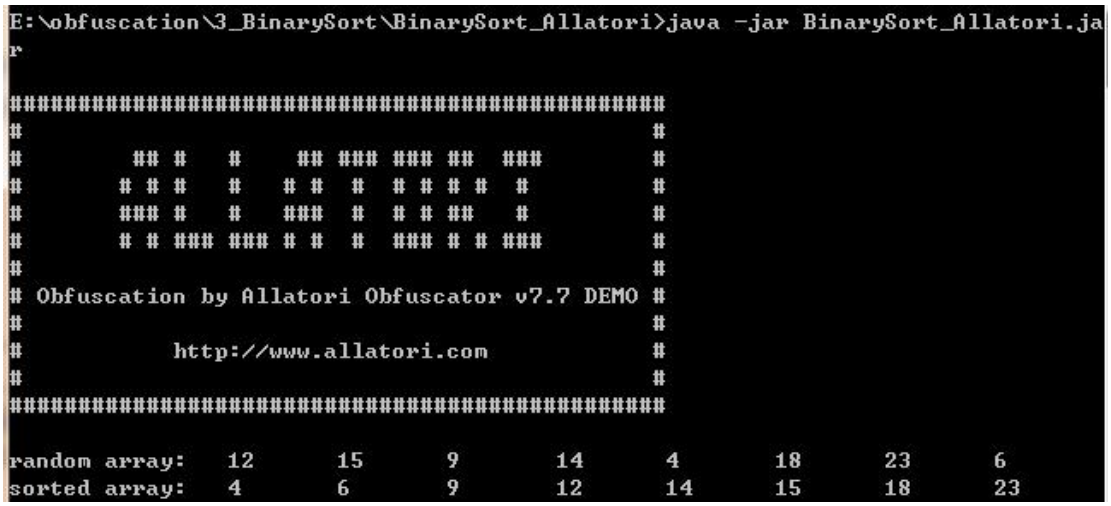

图 6-3 BinarySort.jar 及 BinarySort_Allatori.jar 运行结果

表 7-1 BinarySort.jar 及 BinarySort_Allatori.jar 的正解性

	BinarySort.jar	BinarySort_Allatori.jar
正解性	√	√

从图 6-3 可以看出，Allatori 混淆后的能正确进行排序。

2.3.2 强度

（1）程序长度

使用 SandMark 测试 BinarySort.jar 及 BinarySort_Allatori.jar 的程序长度，结果如表 7-2 所示。

表 7-2 BinarySort.jar 及 BinarySort_Allatori.jar 程序长度

BinarySort.jar 函数名	BinarySort_Allatori.jar 函数名	BinarySort.jar 程序长度	BinarySort_Allatori.jar 程序长度
binarySort	A	70	70
printArray	ALLATORIxDEMO	21	22
main	main	20	23
总计		111	115

（2）控制流循环复杂度

使用 SandMark 测试 BinarySort.jar 及 BinarySort_Allatori.jar 的控制流循环复杂度，结果如表 7-3 所示。

表 7-3 BinarySort.jar 及 BinarySort_Allatori.jar 控制流循环复杂度

BinarySort.jar 函数名	BinarySort_Allatori.jar 函数名	BinarySort.jar 控制流循环复杂度	BinarySort_Allatori.jar 控制流循环复杂度
binarySort	A	4	4
printArray	ALLATORIxDEMO	1	1
main	main	0	0
总计		5	5

使用 Soot 测试 BinarySort.jar 及 BinarySort_Allatori.jar 的控制流循环复杂度，结果如表所示。

表 BinarySort.jar 及 BinarySort_Allatori.jar 控制流循环复杂度

BinarySort.jar 函数名	BinarySort_Allatori.jar 函数名	BinarySort.jar 控制流循环复杂度	BinarySort_Allatori.jar 控制流循环复杂度
binarySort	A	5	5
printArray	ALLATORIxDEMO	2	2

main	main	1	1
总计		8	8

（3）嵌套复杂度

使用 SandMark 测试 BinarySort.jar 及 BinarySort_Allatori.jar 的嵌套复杂度，具体如表 7-4 所示。

表 7-4 BinarySort.jar 及 BinarySort_Allatori.jar 嵌套复杂度

BinarySort.jar 函数名	BinarySort_Allatori.jar 函数名	BinarySort.jar 嵌套复杂度	BinarySort_Allatori.jar 嵌套复杂度
binarySort	A	70	70
printArray	ALLATORIxDEMO	21	22
main	main	20	23
总计		111	115

（4）数据结构复杂度

使用 SandMark 测试 QuickSort.jar 与 QuickSort_Allatori.jar 的嵌套复杂度，具体如表 7-5 所示。

表 7-5 QuickSort.jar 与 QuickSort_Allatori.jar 数据结构复杂度

BinarySort.jar 函数名	BinarySort_Allatori.jar 函数名	QuickSort.jar 数据结构复杂度	QuickSort_Allatori.jar 数据结构复杂度
binarySort	A	19	19
printArray	ALLATORIxDEMO	9	9
main	main	7	7
总计		35	35

2.3.3 执行代价

（1）程序大小

表 7-6 BinarySort.jar 与 BinarySort_Allatori.jar 程序大小

	BinarySort.jar	BinarySort_Allatori.jar（含版本信息）
程序大小（字节）	1228	1793

（2）运行时间

表 7-7 BinarySort.jar 与 BinarySort_Allatori.jar 运行时间

运行时间 ms	run1	run2	run3	run4	run5	run6	run7	run8	run9	run10	avg
BinarySort.jar	31	31	16	16	32	15	16	31	31	31	25
BinarySort_Allatori.jar	31	31	31	16	32	16	15	32	16	16	23.6

2.3.4 隐蔽性

使用 SandMark 测试 BinarySort.jar 与 BinarySort _Allatori.jar 的相似性。比较结果如 7-8 表所示。

表 7-8 BinarySort.jar 与 BinarySort _Allatori.jar 的相似性

BinarySort.jar 函数名	BinarySort _Allatori.jar 函数名	BinarySort.jar 与 BinarySort _Allatori.jar 相似性
binarySort	A	75.7%
printArray	ALLATORIxDEMO	70.4%
main	main	78.9%
Avg similarity		75%