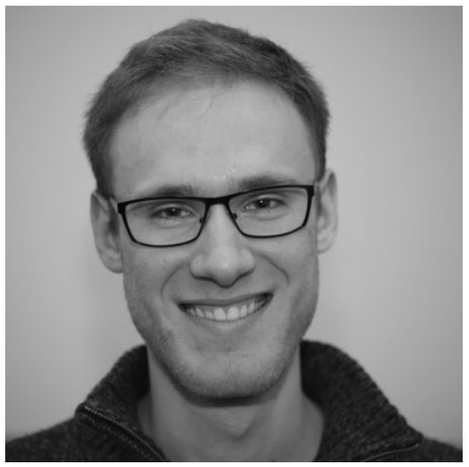
Badger: Complexity Analysis with Fuzzing and Symbolic Execution



Yannic Noller

Rody Kersten

Corina S. Pasareanu

HUMBOLDT-UNIVERSITÄT ZU BERLIN

SYNOPSYS®

Carnegie Mellon University

NASA Ames Research Center

Complexity Analysis

discover vulnerabilities related to worst-case time/ space complexity, e.g., Denial-of-Service

```
0   public void sort (int[] a) {
1       int N = a.length;
2       for (int i = 1; i < N; i++) {
3           int j = i - 1;
4           int x = a[i];
5           while ((j >= 0) && (a[j] > x)) {
6               a[j + 1] = a[j];
7               j--;
8           }
9           a[j + 1] = x;
10      }
11  }
```

Insertion Sort

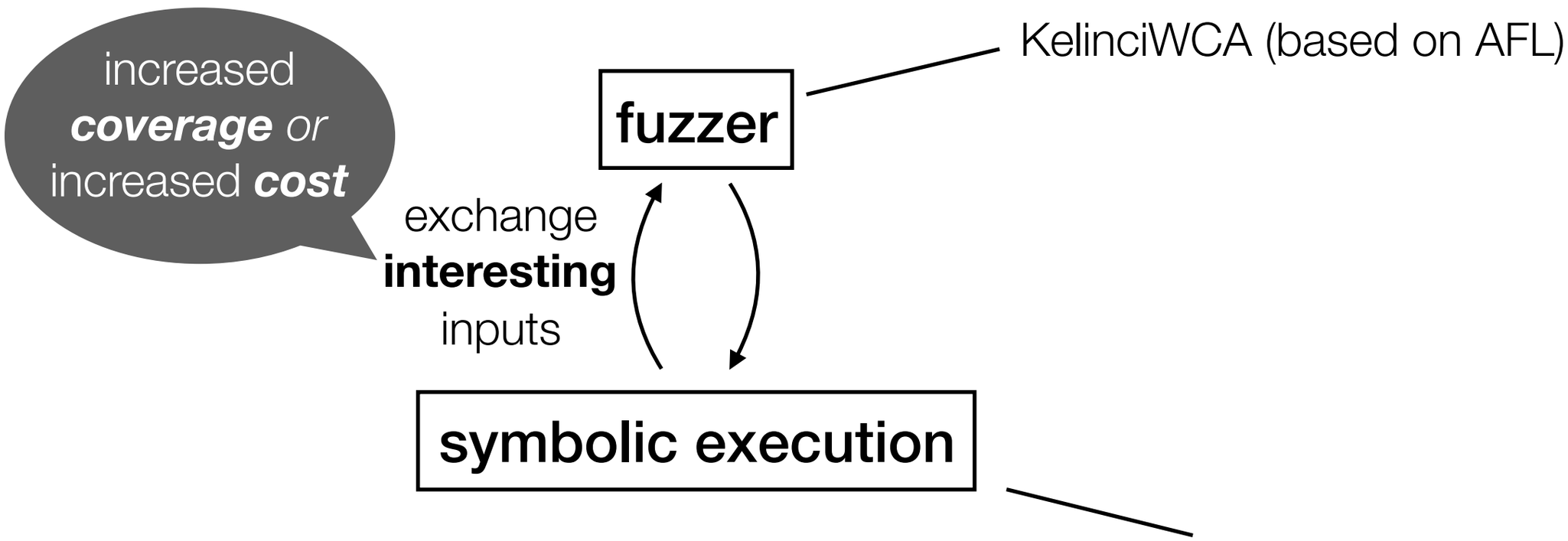
→ find worst-case input:
automated + fast + concrete

- worst-case complexity: $O(n^2)$
- e.g. a=[8, 7, 6] (n=3)

Our Contributions

- combine **fuzzing** and **symbolic execution** to find algorithmic complexity vulnerabilities
- Badger, a framework for analysis of Java applications
- analysis parameterized by a cost metric
- handling of user-defined cost

Badger

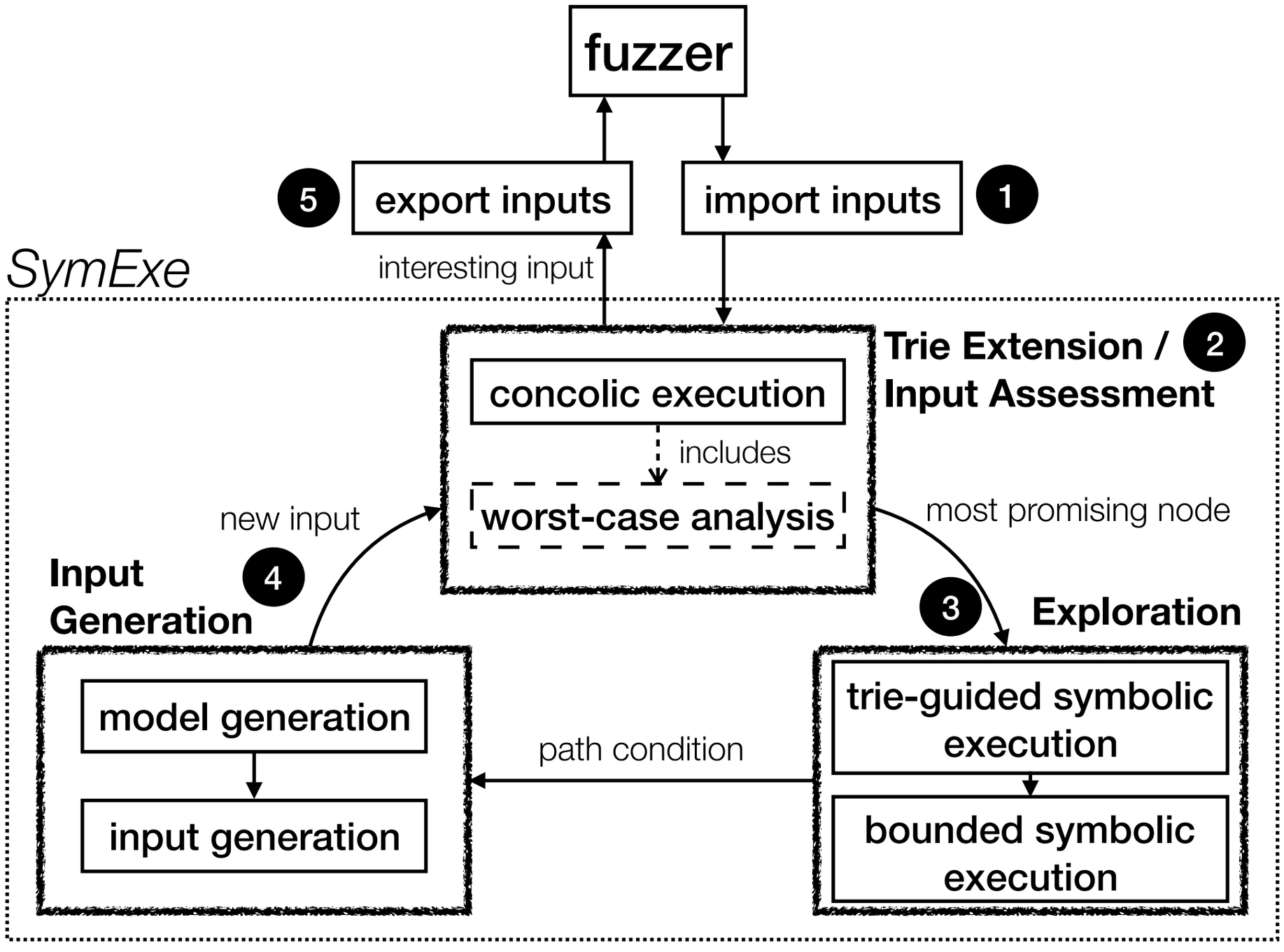


fuzzer and symbolic execution run
in **parallel**

KelinciWCA

- based on AFL, extends Kelinci [Kersten2017]
- mutation-based greybox fuzzing
- cost-guided fuzzer: coverage + cost
- cost metrics: timing / memory / user-defined
- maintain current highscore

SymExe with SPF



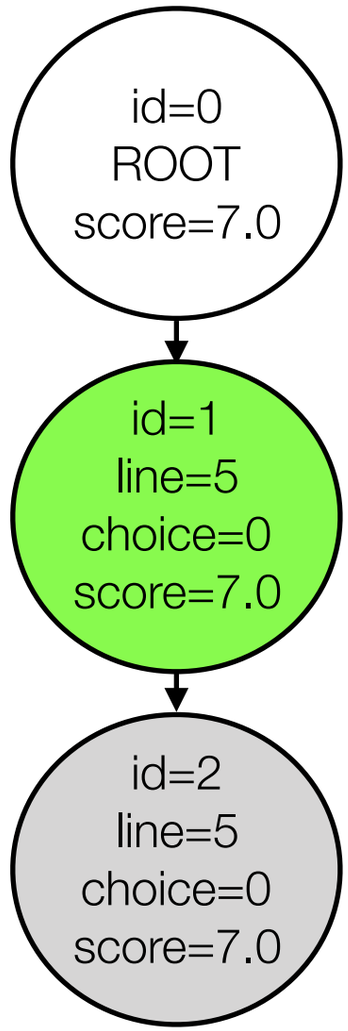
Example

```
0   public void sort (int[] a) {
1       int N = a.length;
2       for (int i = 1; i < N; i++) {
3           int j = i - 1;
4           int x = a[i];
5           while ((j >= 0) && (a[j] > x)) {
6               a[j + 1] = a[j];
7               j--;
8           }
9           a[j + 1] = x;
10      }
11  }
```

Insertion Sort

Trie **extension** with initial input. The **most promising** node get selected.

initial input
a=[37, 42, 48]

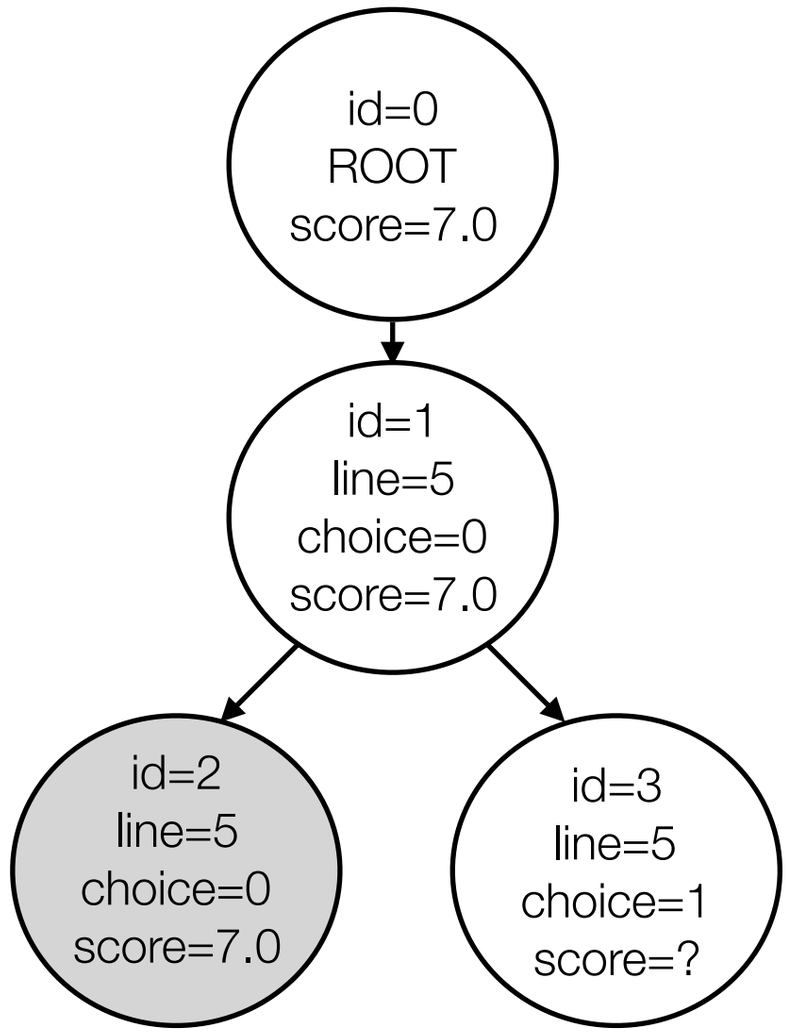


Example

```
0   public void sort (int[] a) {
1       int N = a.length;
2       for (int i = 1; i < N; i++) {
3           int j = i - 1;
4           int x = a[i];
5           while ((j >= 0) && (a[j] > x)) {
6               a[j + 1] = a[j];
7               j--;
8           }
9           a[j + 1] = x;
10      }
11  }
```

Insertion Sort

Exploration and **input generation**.



$$pc = sym\_0 \leq sym\_1 \wedge sym\_1 > sym\_2$$

new input
a=[0, 1, 0]

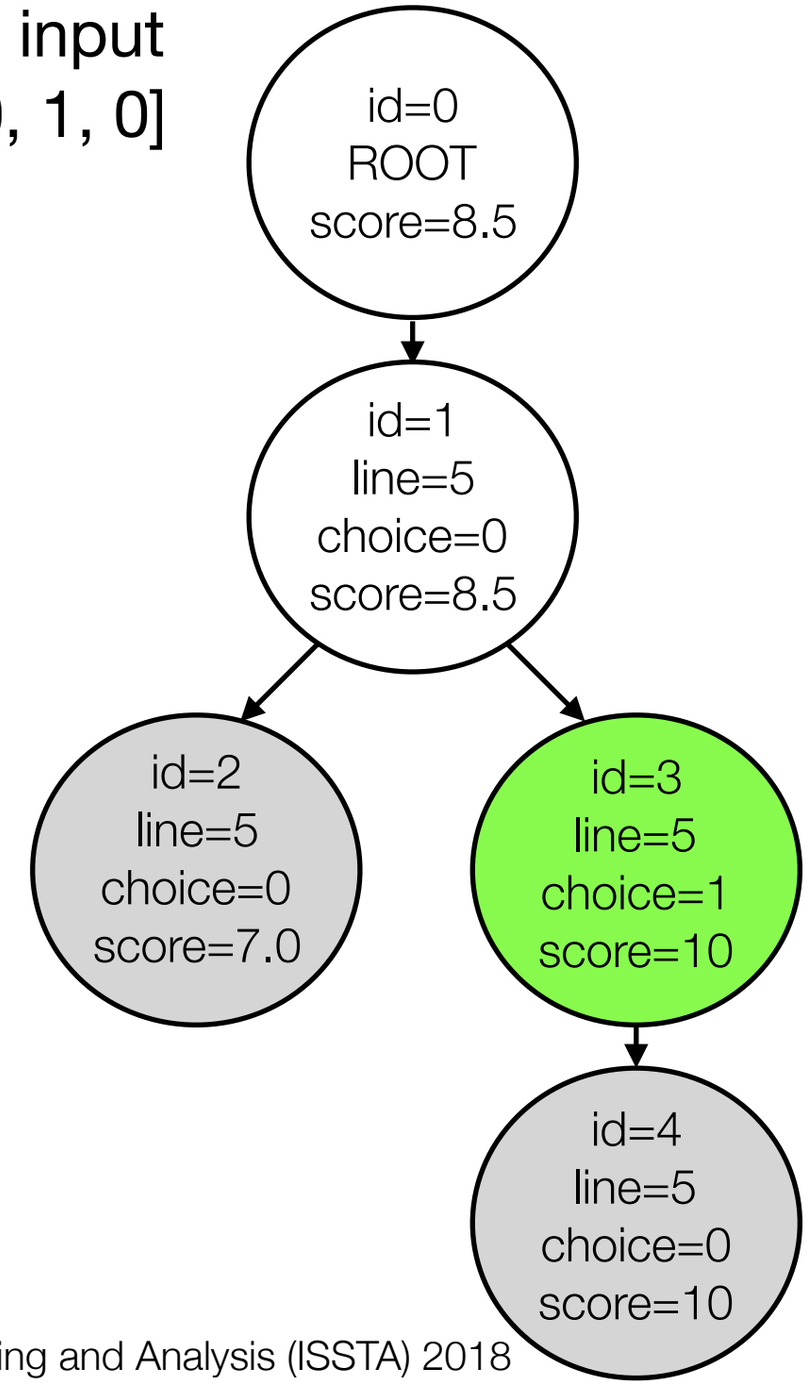
Example

```
0   public void sort (int[] a) {
1       int N = a.length;
2       for (int i = 1; i < N; i++) {
3           int j = i - 1;
4           int x = a[i];
5           while ((j >= 0) && (a[j] > x)) {
6               a[j + 1] = a[j];
7               j--;
8           }
9           a[j + 1] = x;
10      }
11  }
```

Insertion Sort

Assessment of new input and **extension** of the trie. New **most promising** node gets selected.

new input
a=[0, 1, 0]



Research Questions

RQ1: Since Badger combines fuzzing and symbolic execution, is it better than each part on their own in terms of:

(a) Quality of worst-case, and

(b) Speed?

RQ2: Is KelinciWCA better than Kelinci in terms of:

(a) Quality of worst-case, and

(b) Speed?

RQ3: Can Badger reveal worst-case vulnerabilities?

Experiments

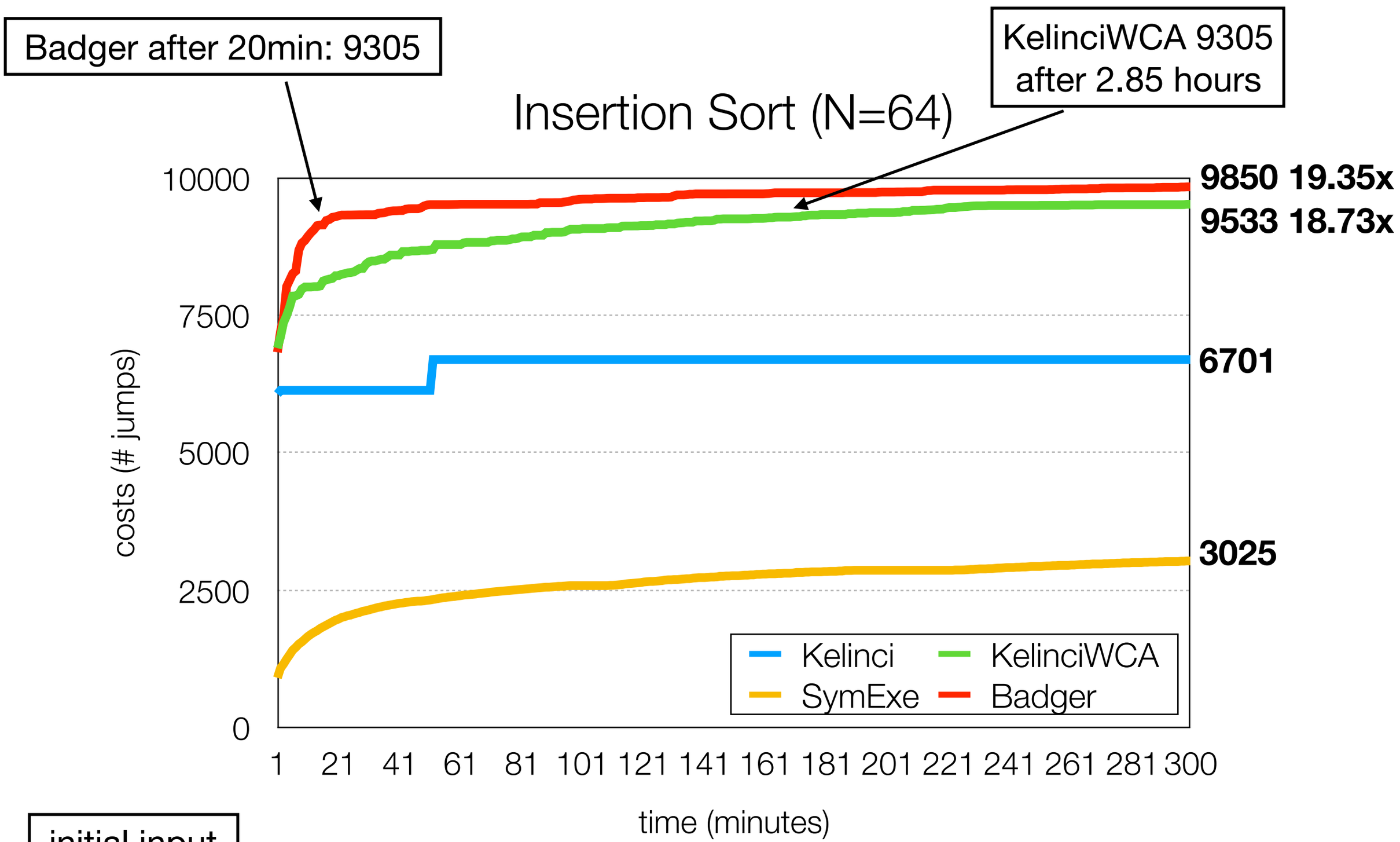
ID	Subject
1	Insertion Sort
2	Quicksort
3a	Regular Expression (fixed input)
3b	Regular Expression (fixed regex)
4	Hash Table
5	Compression
6	Image Processor
7	Smart Contract

each experiment for 5 hours and 5 times

we report the average values (our full data set is available online)

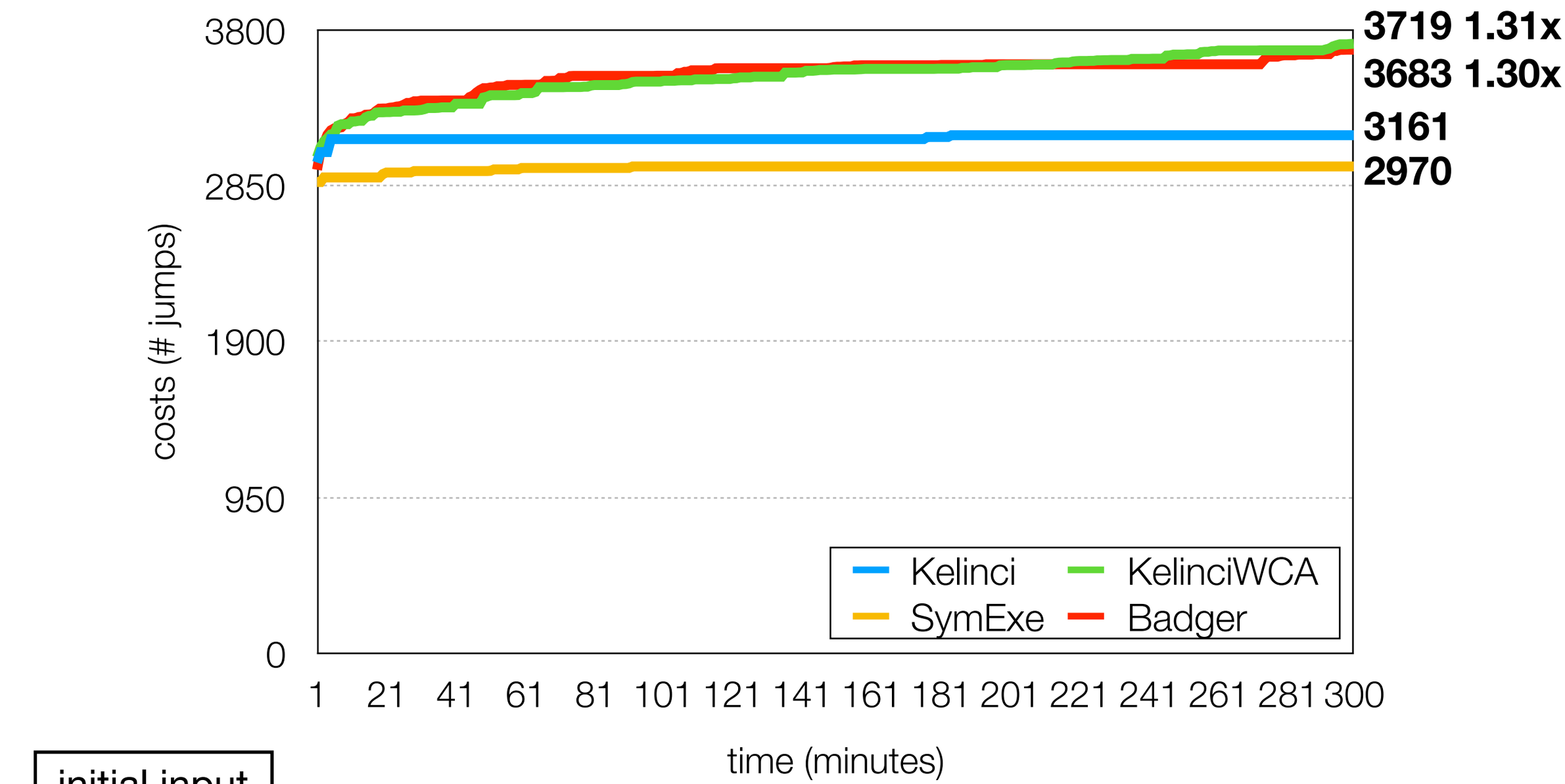
Badger after 20min: 9305

KelinciWCA 9305 after 2.85 hours

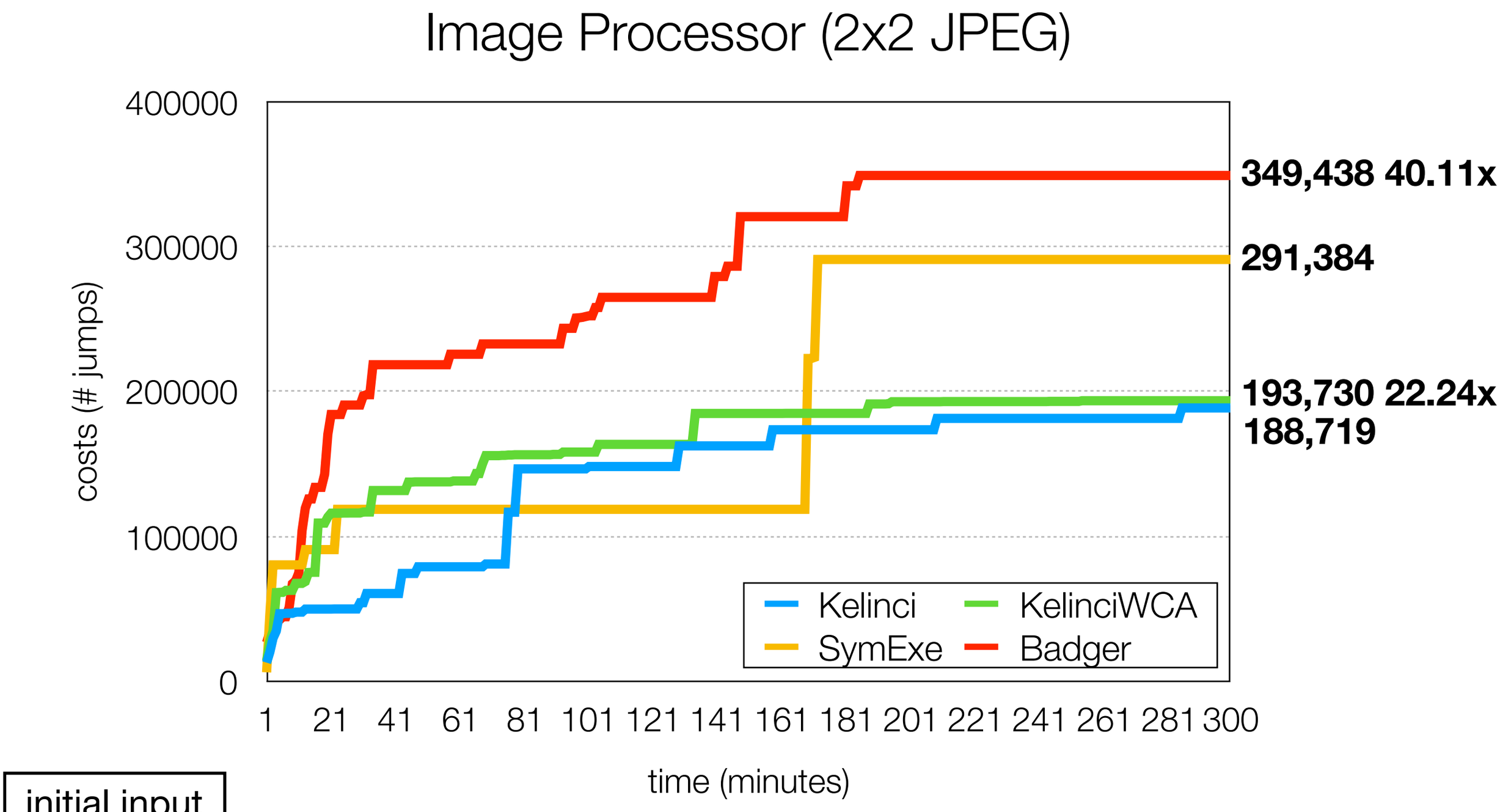


initial input score: 509

no significant difference between Badger and KelinciWCA



initial input score: 2829



initial input
score: 8712

Existing Solutions

- **Fuzzing**
  e.g. SlowFuzz [Petsios2017]
- **Symbolic Execution**
  e.g. WISE [Burnim2009] , SPF-WCA [Luckow2017]
- **Fuzzing + Symbolic Execution**
  e.g. Driller [Stephens2016]

Badger: Complexity Analysis with Fuzzing and Symbolic Execution

Complexity Analysis

discover vulnerabilities related to worst-case time/ space complexity, e.g., Denial-of-Service

```
0  public void sort (int[] a) {
1      int N = a.length;
2      for (int i = 1; i < N; i++) {
3          int j = i - 1;
4          int x = a[i];
5          while ((j >= 0) && (a[j] > x)) {
6              a[j + 1] = a[j];
7              j--;
8          }
9          a[j + 1] = x;
10     }
11 }
```

Insertion Sort

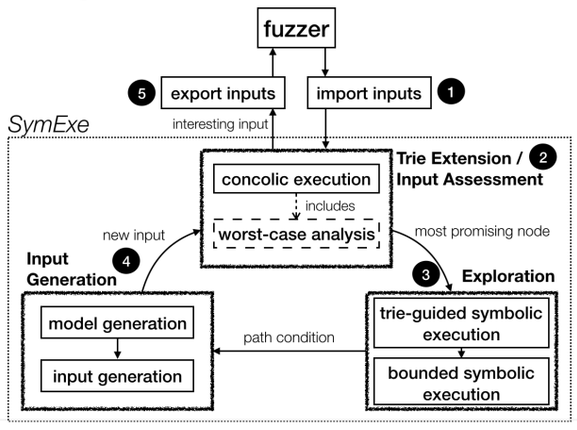
→ find worst-case input: automated + fast + concrete

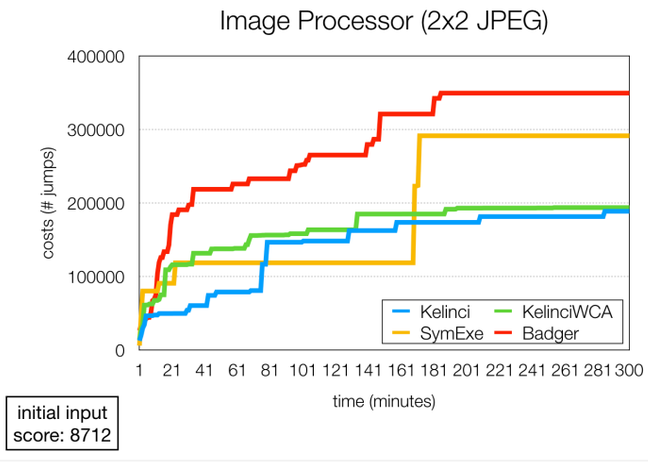
- worst-case complexity: $O(n^2)$
- e.g. a=[8, 7, 6] (n=3)

KelinciWCA

- based on Kelinci [Kersten2017]
- mutation-based greybox fuzzing
- cost-guided fuzzer: coverage + cost
- cost metrics: timing / memory / user-defined
- maintain current highscore

SymExe with SPF





`git clone https://github.com/isstac/badger.git`

References

[AFL] Website. american fuzzy lop (AFL). http://lcamtuf.coredump.cx/afl/.

[Burnim2009] J. Burnim, S. Juvekar, and K. Sen. 2009. WISE: Automated test generation for worst-case complexity. In 2009 IEEE 31st International Conference on Software Engineering. 463–473.

[Clarke1976] L. A. Clarke, "A System to Generate Test Data and Symbolically Execute Programs," in IEEE Transactions on Software Engineering, vol. SE-2, no. 3, pp. 215-222, Sept. 1976.

[Godefroid2005] Patrice Godefroid, Nils Klarlund, and Koushik Sen. 2005. DART: directed automated random testing. In Proceedings a the 2005 ACM SIGPLAN conference on Programming language design and implementation (PLDI '05). ACM, New York, NY, USA, 213-223.

[Kersten2017] Rody Kersten, Kasper Luckow, and Corina S. Păsăreanu. 2017. POSTER: AFL-based Fuzzing for Java with Kelinci. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17).

[King1976] James C. King. 1976. Symbolic execution and program testing. Commun. ACM 19, 7 (July 1976), 385-394.

[Luckow2017] Kasper Luckow, Rody Kersten, and Corina Pasareanu. 2017. Symbolic Complexity Analysis using Context-preserving Histories. In Proceedings of the 10th IEEE International Conference on Software Testing, Verification and Validation (ICST 2017). 58–68.

[Miller1990] Barton P. Miller, Louis Fredriksen, and Bryan So. 1990. An empirical study of the reliability of UNIX utilities. Commun. ACM 33, 12 (December 1990), 32-44.

[Petsiois2017] Theofilos Petsios, Jason Zhao, Angelos D. Keromytis, and Suman Jana. 2017. SlowFuzz: Automated Domain-Independent Detection of Algorithmic Complexity Vulnerabilities. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). ACM, New York, NY, USA, 2155-2168.

[Sen2005] Koushik Sen, Darko Marinov, and Gul Agha. 2005. CUTE: a concolic unit testing engine for C. In Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-13). ACM, New York, NY, USA, 263-272.

[Stephens2016] Stephens N, Grosen J, Salls C, Dutcher A, Wang R, Corbetta J, Shoshitaishvili Y, Kruegel C, Vigna G. Driller: Augmenting Fuzzing Through Selective Symbolic Execution. InNDSS 2016 Feb (Vol. 16, pp. 1-16).