An In-Class Activity Exploring Hash Function Quality

Ellen Spertus
e.spertus@northeastern.edu
Northeastern University
Oakland, CA, USA

Rasika Bhalerao
r.bhalerao@northeastern.edu
Northeastern University
Oakland, CA, USA

Course Data Structures
Programming Language Java
Knowledge Unit Data Structures
CS Topics Maps/Hash Tables/Dictionaries, Character/String types
Resource Type Lab

Synopsis

This in-class activity enables students to explore how the quality of a hash function affects the performance of a hash table. Students write their own `hashCode()` function for Java strings and submit it to an autograder that determines the number of collisions that would occur with a given corpus. The results are displayed on a leaderboard for students to see in real time how their implementation compares to other students' and the Java libraries. Afterwards, students are eager to see the winning implementations and the Java library implementations. This activity has been used successfully with 480 students in 10 sections over 3 years on Gradescope.

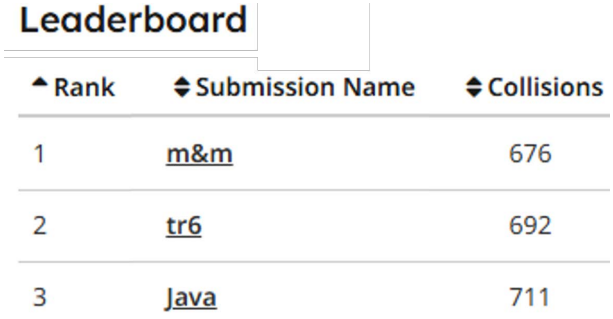
Keywords

Data Structures, Hashing, Hash Collisions, Autograding

ACM Reference Format:
Ellen Spertus and Rasika Bhalerao. April 2026. An In-Class Activity Exploring Hash Function Quality. In *ACM EngageCSEdu.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3801161

1 Engagement Highlights

Hash tables and hash sets seem too good to be true: Not only are they widely applicable and easy to use, but they have optimal asymptotic performance, with $O(1)$ access time and using $O(n)$ space to store n elements. These amazing characteristics can pique students' curiosity. How can such efficiency be achieved? The answer depends not only on the implementation of the hash table but also on the hash function. This in-class activity invites students to write their own implementation of `hashCode()` for a string-like Java class, which they can do individually or in teams. After testing the implementation locally, students submit it with a pseudonym to an autograder that simulates hashing a corpus of strings and displays the number of collisions on a leaderboard (Figure 1). Students engage in spirited collaboration and competition as they come up with creative approaches. Afterwards, they are eager to see the winning implementations.



Leaderboard

Rank	Submission Name	Collisions
1	m&m	676
2	tr6	692
3	Java	711

Figure 1: Gradescope leaderboard showing the rank, pseudonym, and number of collisions.

The activity applies several NCWIT engagement practices:

- **Use Meaningful and Relevant Content** Our experience is that students are interested in how hash tables perform so well, which this elucidates. While the provided activity is based on Java strings, it can be adapted to use other data types, such as ones that students implemented on previous assignments. After performing the exercise, students are eager to see others' implementations.
- **Incorporate Student Choice** The only restriction placed on implementations is that they must be the students' own work, without the use of AI or other `hashCode()` implementations. There are always a variety of approaches.
- **Give Effective Encouragement** The leaderboard is seeded with a bad implementation, which is easy for students to beat, and implementations from the Java standard libraries, which a few students per section beat, to their pride and delight.

ACM EngageCSEdu, April 2026.

ACM ISBN 979-8-4007-2696-5/2026/04
https://doi.org/10.1145/3801161

- **Offer Student-Centered Assessment** The autograder gives immediate feedback, enabling students to iterate and improve their submissions, promoting a growth mindset. The leaderboard mitigates Imposter Syndrome, enabling students to see how their performance really compares to classmates.

It is important to acknowledge that no activity works equally well for all students. This activity is not ideal for students who prefer to work alone in quiet environments without time constraints. This assignment is not for credit, so there is no penalty for not participating, although almost all students do.

2 Recommendations

2.1 Background Knowledge

To get the most out of the exercise, students should know how the `hashCode()` method is used by implementations of hash tables (i.e., when adding or retrieving a key-value pair or testing if a key is present) and what the requirements are for the `hashCode()` function. We recommend sharing or summarizing the specifications in the Java API and in *Effective Java* by Joshua Bloch [1]. For objects in which equality depends only on the values of immutable instance variables, the rules can be summarized:

- When called repeatedly on the same object, the same value must be returned.
- When called on equal objects (according to the type's `equals()` method), the same values must be returned.

We like to ask students whether this implementation, provided by Bloch, is legal:

```
@Override
public int hashCode() {
    return 42;
}
```

As Bloch explains:

> It's legal because it ensures that equal objects have the same hash code. It's atrocious because it ensures that every object has the same hash code. Therefore, every object hashes to the same bucket, and hash tables degenerate to linked lists.

We also present and discuss the limitations of two simple approaches:

- returning the length of the string
- returning the numeric value of the first character of the string

These have the benefit of providing students with methods likely to be of use in their implementations.

2.2 Contest

The supplementary material includes detailed instructions for setting up a Gradescope autograder, which requires an Institutional account. We recommend seeding the autograder with at least these three submissions: Bloch's strawman implementation and the Java 1 and 2+ implementations of `String.hashCode()`.

Before opening the contest, students should be informed that they can specify a pseudonym when uploading their submission so their real name does not appear on the leaderboard, which their classmates can see. Real names are visible only to the instructor. Students will be curious to see winning implementations, so it is good to come up with an agreement about how winning submissions can be shared. If the teacher shows code through the Gradescope website, the submitter's name will be disclosed. Students might be given the option of preceding their pseudonym with a minus sign or underscore to indicate that their code and name should not be shared.

After distributing the code and instructions, we leave the contest open for about ten minutes. During this time, we display the leaderboard on the screen and walk around the classroom, encouraging and helping students.

2.3 Discussion

After the contest (during the same class period or the next), we share winning submissions and the implementations of String's `hashCode()` method from Java 1 and later versions. Remarkably, the Java 1 implementation [2] takes $O(1)$ time — independent of the length of the string. It was replaced in Java 2 and has remained unchanged since then.

3 Materials

- `README.md`, instructions for presenting the material, including setting up a Gradescope autograder and a link to GitHub Classroom instructions
- `presentation/`, a directory with (1) a student-facing PowerPoint presentation, (2) Java files to share with students, and (3) sample submissions to seed the leaderboard
- `autograder.zip`, an autograder that can be uploaded to Gradescope
- `autograder-src.zip`, files used to create the autograder, in case modifications are desired

References

[1] Joshua Bloch. 2017. *Effective Java* (3rd ed.). Addison-Wesley Professional.

[2] Sun Microsystems. 2001. *String.java.* Retrieved March 22, 2026 from https://github.com/barismeral/Java-JDK-1.0-src/blob/master/src/java/lang/String.java