CSM 61B
Spring 2018

# Hashing Crib Sheet

March 17, 2018

## 1 Overview

The `hashCode()` method maps an object to an integer and needs the following in order to be valid:

- Consistency: when `hashCode()` is called on the same object multiple times, the same value should be returned.
- Equality: if `o1.equals(o2) == true`, then `hashCode(o1)` should be equal to `hashCode(o2)`.

Different objects don't necessarily need to have different hashCodes, but it is better if they do. Further, if the `hashCode()` method is not overridden, the default `Object.hashCode()` is used, which returns the address of the object.

## 2 Implementation

The underlying implementation of a `HashSet` is an array of linked lists. Elements in the `HashSet` are placed in the linked list at `index = Math.abs(hashCode) % array.length` of the array.*

Consider each linked list to be a "bucket" of elements; each bucket is limited to capacity $m$. The load factor is defined to be $\alpha = \frac{n}{m}$, where $n$ is the total number of elements. When any bucket exceeds the load factor, the array is resized and elements are redistributed.

*Note: You may see `index = (hashCode & 0x7fffffff) % array.length` being used instead. This ensures that the resulting number is nonnegative and does not necessarily produce the same result as `Math.abs(hashCode) % array.length`. The absolute value computation will be used in the following explanations.

## 3 Operations & Runtimes

1. `add(E e)`: $\Theta(1)$
   - Try to add element `e` to the bucket at `index = Math.abs(e.hashCode()) % array.length`.
   - If the load factor is exceeded, resize by doubling the size of the array. Note that `array.length` has changed, so you must redistribute and recompute the indices of each element. For each element $e$ in HashSet, add $e$ to the bucket at its corresponding `index`.

   Typically, it takes $\Theta(1)$ time to add an element to the HashSet. Occasionally, the load factor will be exceeded and a resize will occur; this takes $\Theta(N)$ time.

2. `contains(E e)`: $\Theta(1)$

   - Check if element `e` is in bucket at `index = Math.abs(e.hashCode()) % array.length`.

   Bucket size is limited by the load factor, which is a constant. Normally, if the hash function is good and elements are distributed evenly across buckets, it takes $\Theta(1)$ time to perform this check. If the hash function is bad, however, and all elements are in the same bucket, `contains` can take $\Theta(N)$ time.