# Lecture 25: Maps & Dictionaries

## CS 62

Fall 2017

Kim Bruce & Alexandra Papoutsaki

# Maps

- Collection of associations between a key and associated value
- Store and retrieve data based on a key.
  - Store phone numbers by name.
  - Store word pair frequencies by first word.
  - Store account info by user ID.

- At most one value per key
(matches the mathematical concept).
- Also known as “dictionaries”, “symbol tables” or “associative arrays”.

# Interface

```
public interface Map<K,V> {
   int size();
   V get(Object key);
   V put(K key, V value);
   V remove(Object key);




}
```

# Interface

```
public interface Map<K,V> {
  int size();
  V get(Object key);
  V put(K key, V value);
  V remove(Object key);

  boolean containsKey(Object key);
  boolean containsValue(Object value);
  Set<K> keySet();
  Collection<V> values();
}
```

# Implementations

| Data Structure | search | insert | delete |
|---|---|---|---|
| List | $O(n)$ | $O(1)$ | $O(n)$ |
| Sorted list | $O(logn)$ | $O(n)$ | $O(n)$ |
| Balanced BST | $O(logn)$ | $O(logn)$ | $O(logn)$ |
| Array[“key range”] | $O(1)$ | $O(1)$ | $O(1)$ |

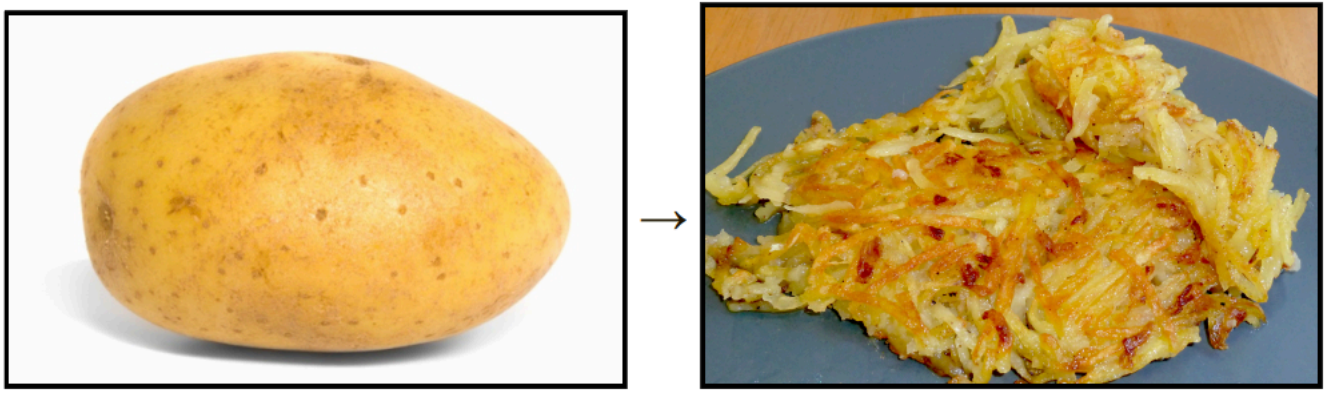
Last row is array where keys are subscripts

`http://bigocheatsheet.com/`

# Problem

- Goal: Array-like performance for all keys
- Problems:
  - Keys are not integers
    (and there is no obvious way to convert them)
  - Key range may be large or infinite
    (and keys may be sparse)
    - Suppose use SS#'s as subscripts to table of students

# Hashing



Instead provide function from keys to subscripts that is denser.

# Perfect Hashing

```
int hash(Object o);
```

- Should be $O(1)$.
- Should return an integer.
- The integers for our $n$ keys should be $0$ ... $n$-$1$.
- Must be a unique integer for every object.
  - That is, it should be bijective.
- Given hash, just use an array where: items[H(key)] = value

- So important that hashCode function built-in to Java classes.

# Hash Functions

- Look for reasonable function that scatters elements through array randomly so won't bump into each other.
- Lose any ordering on keys
- Ideal is to find in time $O(1)$.
- We want to:
  - Find good hashing functions
  - Figure out what to do if 2 elements are sent to same location

- *"A given hash function must always be tried on real data in order to find out whether it is effective or not."*

# Actual Hashing

- Unique integer for an Object?
  Its address in memory.

- Numbers in $0 \ldots n\text{-}1$?
  Take the modulus by $n$

```
public int hash(Object o, int n) {
  return addr(o) % n;
}
```

# Actual Hashing

- ✓ Should be O(1)
- ✓ Should return an integer.
- ✓ The integers for our $n$ keys should be $0 \ldots n\text{-}1$.
- X Must be a unique integer for every object.
  (true in the limit as $n \to \infty$ )

```
public int hash(Object o, int n) {
  return addr(o) % n;
}
```

# Actual Hashing

- Call `obj.hashCode` instead of `hash(obj)`
- Let each map object do the modulus ($n$ is different)

```
public int hashCode() {
   return addr(this);
}
```

# Handling and Equality

```
public class Point {
  public int x, y;

  public boolean equals(Object other) {
    if (other instanceof Point) {
      return (this.x == other.x
           && this.y == other.y);
    }
    return false;
  }

  public int hashCode() { return addr(this); }
}
```

# Problems

- What to do when results aren't unique?
- What about objects with `.equals`?
- How can we get a good distribution of results?