



---

## Hash Sets, Bags, and Tables

Wellesley College CS230  
Lecture 25  
Thursday, May 3  
Handout #43

Final Project Phase 3 (Program Skeleton) due Monday, May 7

20 - 1



---

## Overview of Today's Lecture

- Balanced search trees give  $\Theta(\log(n))$  worst-case search, insertion, deletion times for sets, bags, tables.
  - (1) Can we do better? Yes - for average (not worst) case
  - (2) Can we get by without Comparators? Yes!
- A simple example: integer sets. Key ideas = clever array indexing and time/space tradeoff.
- Hashing = a way of viewing any object as an integer.
- Our own hash sets, bags, and tables
- Java's hash sets and tables

20 - 2

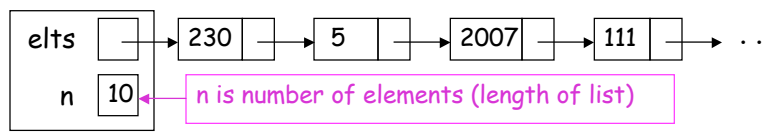


## Motivating Example: Integer Sets

Suppose we want to represent sets of integers in the range  $[0 .. \text{Integer.MAX\_INT}]$ .

Example:  $\{2, 5, 7, 111, 112, 230, 231, 249, 251, 2833049\}$

Representation 1: unordered list without duplicates



This uses one list node per element (so is space-efficient), but search, insertion, and deletion all take  $\Theta(n)$  time.

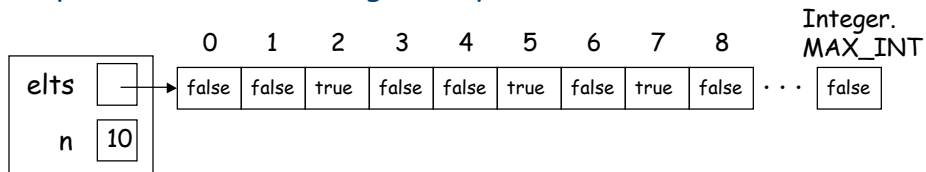
20 - 3



## Integer Sets: A Time-Efficient Representation

Example:  $\{2, 5, 7, 111, 112, 230, 231, 249, 251, 2833049\}$

Representation 2: A large array of booleans



Now search, insertion, and deletion take  $\Theta(1)$  time ...

... but we need lots of space even for a small set.

This exemplifies a **time/space tradeoff**: we often can get faster running times if we're willing to use more space.

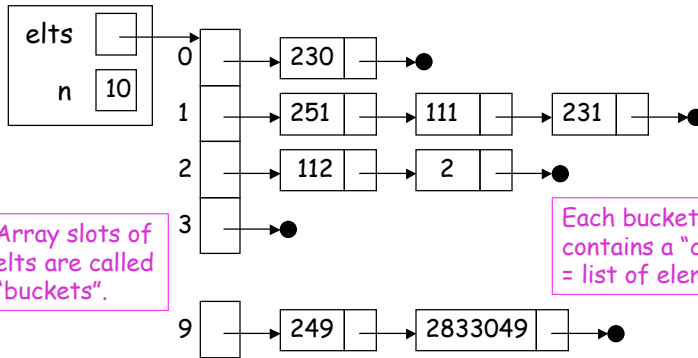
20 - 4



## Integer Sets: A Happy Medium?

Example: {2, 5, 7, 111, 112, 230, 231, 249, 251, 2833049}

Representation 3: A smaller array of unordered lists indexed by (integer-value mod array-size)



Array slots of elts are called "buckets".

Each bucket contains a "chain" = list of elements

We happened to make elts.length = 10 = n, but could make elts any size.

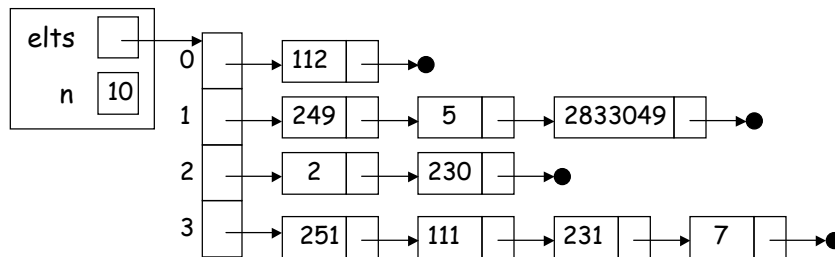
20 - 5



## Can Use Any Number of Buckets

Example: {2, 5, 7, 111, 112, 230, 231, 249, 251, 2833049}

elts.length = 4



20 - 6



## Load Factor

load factor  $\alpha = n / \text{elts.length} = \text{average chain length}$

n	elts.length	$\alpha$
10	4	2.5
10	8	1.25
10	10	1
10	100	0.1
50	100	0.5

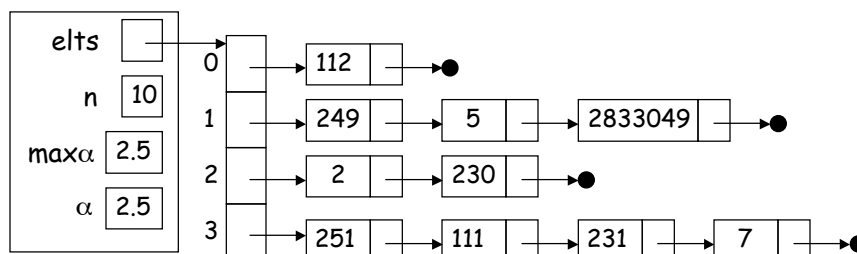
Searching, insertion, deletion times are:

- $\Theta(n)$  in the worst case (all elements in same bucket)
  - $\Theta(\alpha)$  in the average case:
    - If elts.length is held constant, then  $\Theta(\alpha) = \Theta(n)$
    - If  $\alpha$  is held constant, then  $\Theta(\alpha) = \Theta(1)$
- Limit  $\alpha$  by increasing elts.length to keep it  $\leq$  preset threshold 20 - 7



## Resizing Example: Before

Example: {2, 5, 7, 111, 112, 230, 231, 249, 251, 2833049}



Suppose we insert 173.

If didn't change elts.length,  $\alpha$  would become  $11/4 = 2.75 > \text{max}\alpha$ .

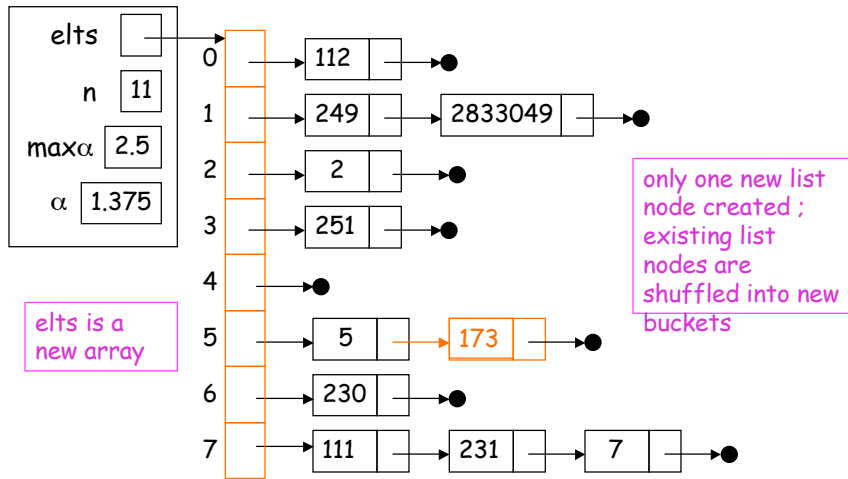
So we'll double elts.length and reinsert all elements in new array.

20 - 8



## Resizing Example: After

Example: {2, 5, 7, 111, 112, 230, 231, 249, 251, 2833049}



20 - 9



## Hashing: Viewing any Object as an Integer

We can generalize the integer set example to sets of objects of any type T for which we can convert an instance to an integer.

In Java, every Object has the instance method `public int hashCode();`

object	hashCode()	object	hashCode()
"a"	97	"Java programs"	1515216525
"b"	98	new Integer(230)	230
"ant"	96743	new Character('a')	97
"bat"	97301	new Point(1,2)	-1048576
"cat"	98262	new Point(2,1)	-32505856
"dog"	99644	new Point(0,0)	0
"beetle"	-1392911061		

20 10



## Some notes on Java Hashing

---

Objects that are `.equals()` must have the same `.hashCode()`

Objects with the same `.hashCode()` may or may not be `.equals()`

Programmers can override default `.hashCode()`

Must be careful when using mutable objects as keys!  
(E.g. Point instances)

20 - 11



## Hashed Structures in Standard Java

---

```
public class Hashtable<K,V>
```

```
public class HashMap<K,V>
```

```
public class HashSet<E>
```

20 - 12