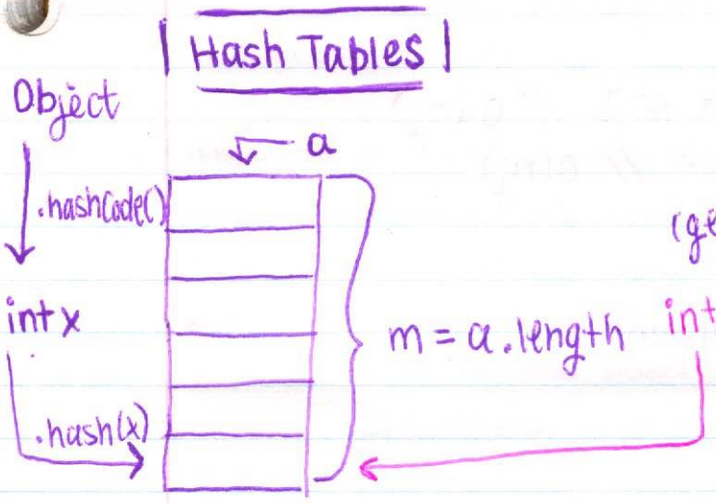
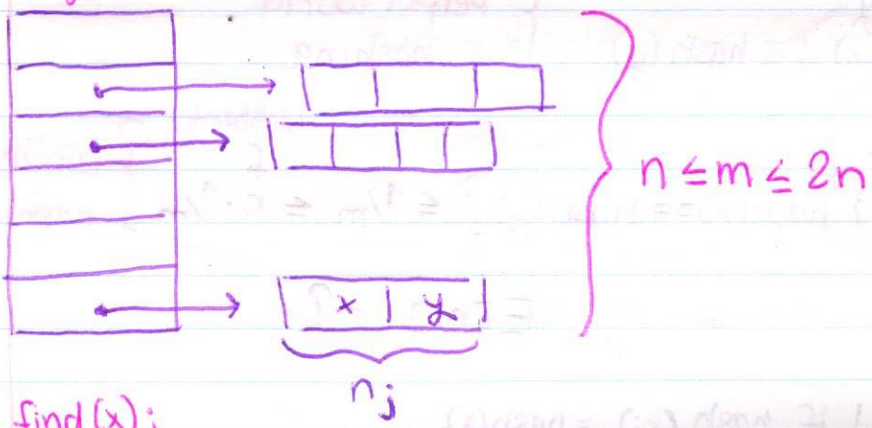
Nov 1st
COMP 2402

Hash Tables



→ x goes in the array somewhere

Chaining (collision prevention




```
find(x);
  list = a[hash(x)]      // O(1)
  search the list for x  // O(n_j)

add(x);
  find(x)                // O(n_j)
  if not found:
      a[hash(x)].add(x)  // O(1)
  if(full) resize();     // O(1) amortized
```

```
remove(x);
    iterate over a[hash(x)]   // O(n_j)
    remove if found           // O(n_j)
```

```
hash(x):
    return 0;
```
} n collisions

```
hash(x);
    return int(Math.random().m)
```

if $x != y$
$hash(x) != hash(y)$ } Perfect world hashing

if $x != y$
$Prob\{hash(x) == hash(y)\} \leq 1/m \leq c \cdot 1/m$ } universal hashing
(c: constant)

Expected value of n_j $E[n_j] = ?$

$$I_i = \begin{cases} 1 & \text{if } hash(x_i) = hash(x) \\ 0 & \text{otherwise} \end{cases}$$

$$n_j = \sum_{i=1}^{n} I_i$$

$$E\left[\sum_{i=1}^{n} I_i\right]$$

$$= \sum_{i=1}^{n} E[I_i]$$

$$= \sum_{i=1}^{n} Pr\{I_i = 1\}$$

$$\leq \sum_{i=1}^{n} c/m$$

$$= n \cdot c/m \quad // \; n/m \leq 1$$

$$\leq c$$

Multiplicative Hashing

$hash(x) = ((r \cdot x) \bmod 2^w) \operatorname{div} 2^{w-d}$

r = static random odd number in $\{0, \ldots, 2^{w-1}\}$
w = wordsize of numbers (32-bits in java)
d = dimension of the hash table ($m = 2^d$)

$x = \underbrace{1011011 \ldots 101}_{w \text{ bits}}$

$r = \underbrace{1011110111 \ldots 1}_{w \text{ bits}} \equiv$ (must be odd)

$x \cdot r \,\%\, 2^w = (x \cdot r)$ // in Java

$x \cdot r = \underbrace{1011011101101 \ldots 1110110110010}_{2 \cdot w \text{ bits}}$ (w bits we're interested in (lowest order))

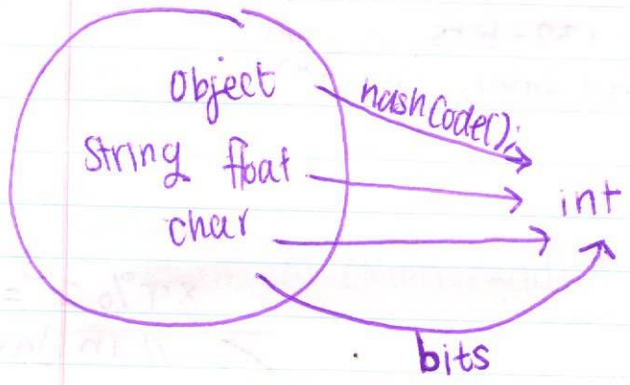
$= \underbrace{1101101101101}_{w \text{ bits}}$

$= \underbrace{1101101101101}_{d \text{ bits}} \underbrace{\ldots 1011}_{w-d \text{ bits}}$

$(x \cdot r) >> w-d$

.hashCode() and .equals()

if you overrride .hashcode() you must change .equals() and vice versa



$h_1, h_2, h_3, \ldots, h_K$

$h_1 r_1 + h_2 r_2 + h_3 r_3 + h_4 r_4 + \ldots + h_K r_K$