Announcements

Programming assignment #2

- Back in a week or two ⇒ Heap
- Min of three function

```
minof3(i, j, k) {
    min = i;
    if(j bounds... A[j] < A[min]) min = j;
    if(k bounds... A[k] < A[min]) min = k;
```

Programming assignment #3 ⇒ Quicksort

- Due tomorrow
- Be sure to format the output
- The constant should be within some range

Programming assignment #4 ⇒ Radix-bucket sort

- Same input and output as quickswap

Dynamic set

Supported operations ⇒ Operations should be O(1) (may be a worst case of O(n)) more often $O(1+\alpha)$

- insert ⇒ add an element from the set
- delete ⇒ remove an element from the set
- search ⇒ is an element in the set

Basic idea

- There is a hashing function which takes in a value to be stored and returns an index
- Elements are placed in an array at the position returned by the hash function
- For integers in range 0...n (where n has some limit) a direct map in which the element is placed in an array as a boolean (true) at the index of its value

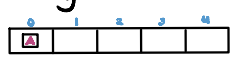
Collisions ⇒ There may be collisions in which multiple elements are hashed to the same index

- A collision requires a solution
- A hashtable with no collisions is a uniform hash (which can be achived on a known data set with math)
- Normally there will be collisions

Collision resolution ⇒ Chaining and open addressing $\alpha = \frac{\text{items in table (n)}}{\text{table size (m)}}$

Chaining

← all same hash, list of elements from that position $\alpha \geq 0$



Open addressing

pick a new spot if taken $\alpha \geq 0.0, \alpha \leq 1.0$

Chaining ⇒ Insert psudo code

```
table[getHash(x)].insert(x);
//Usually will have to account for duplicates (O(1) search)
// For our assignment no testing for duplicates
```

Hashing function ⇒ Three types (m - table size, n - data size)

1) Division method (awful) ⇒ `hash(k) = k % m` //convert k to int if needed
   - × Dependant on table size
   - × Constrains "best" table size selection (prime # not near a power of 2)
2) Multiplication method (better) ⇒ $hash(k) = \lfloor m(kc - \lfloor kc \rfloor) \rfloor$ (% location in table; const; floor (integer portion))
   - ✓ Independant of table size
   - × Hacked if c is known
   - · Choosing c
     - · Ideally analyzie all data, choose c that minimizes collisions
     - * Good value for c without data analysis ⇒ $\frac{1}{\phi} \approx 0.618034$
3) Universal hashing (secure) ⇒ Same as 2, but c is selected at runtime
   - ✓ Independant of table size
   - ✓ Hard to hack (would have to guess c)