Complexity Review

```
Function PosNums(n)
    x ← new ArrayList
    For i In 1 to n:
        x.insertAtTail(i)
    EndFor
    Return x
EndFunction
```

Heaps

insert(P, V)
add (P,V) to end of list
bubble Up

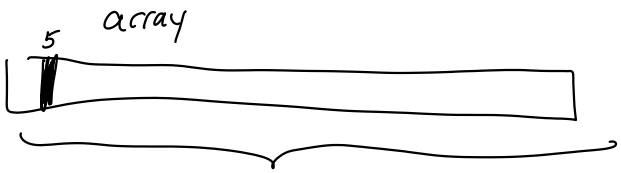
worst case insert $O(n)$
amortized worst case insert $O(\log n)$

What's a dictionary?

ADT of mappings $K \mapsto V$

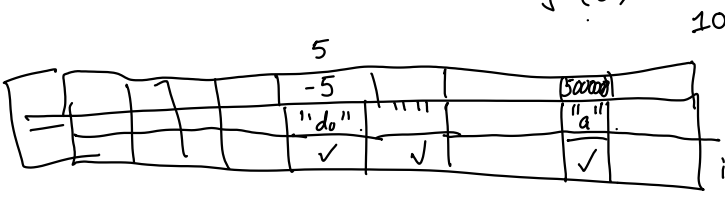
	Balanced BST	Hash Table
get	$O(\log n)$	$O(1)$
insert	$O(\log n)$	$O(1)$
update	$O(\log n)$	$O(1)$
remove	$O(\log n)$	$O(1)$

1. Keys are ints
~~2. Keys are non-negative~~
~~3. Keys are < 1,000,000~~
~~4. No repeat inserts~~



insert(5000008, "a")
get(8)

insert(5, "hi")



insert(-5, "do")

hashing function: takes a value and projects it into a fixed range

```
int hash(int n, int range) {
    n = n % range;
    if (n<0) {
        n += range;
    }
    return n;
}
```

```
int hash(string s, int range) {
    return s.length();
}
```

bad hash

```
int hash(string s, int range){
    return 0;
}
```

WORST HASH

```
int hash(string s, int range) {
    int acc = 0;
    for each c in s {
        acc *= 31;
        acc += c;
    }
    ....
}
```

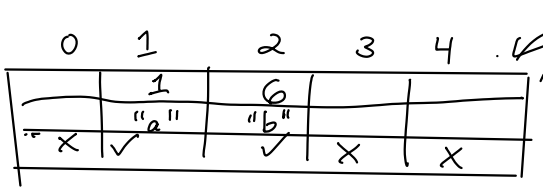
"ba" "ab"

good hash

```
int hash(string s, int range) {
    int acc = 0;
    for (int i=0; i<s.length(); i++) {
        acc += s[i];
    }
    return acc % range;
}
```

okay hash

*



insert (1, "a")
insert (6, "b")
get (6)
get (16)

```
insert(K key, V value)
    i ← hash(key, capacity)
    while (contents[i].inUse && contents[i].key ≠ key) {
        i++;
        i ← i % capacity;
    }
    if (contents[i].key == key): //
    contents[i].key ← key;
    contents[i].value ← value;
    contents[i].inUse ← true;
```

Linear Probing Hashtable

collisions are handled by using a neighboring position

All operations are average O(1) if hash function distributes inputs over the output range evenly

Hashtables: "expandCapacity"

$$\text{load factor} = \frac{\text{number of mappings}}{\text{capacity}}$$