Faster and faster and … search

● Binary search trees
  ➢ average case insert/search/delete = $O(\quad)$
  ➢ worst case = $O(\quad)$

● balanced search trees exist, worst case is good
  ➢ AVL trees, red-black trees, 2-3 or 2-3-4 or B-trees
  ➢ why not always use balanced trees? When?

● Comparison based search: $\Omega(\log n)$
  ➢ lower bound on how fast we can search
Beating lower bounds

- How can we prove the lower bound?
  - Distinguishing between different keys: binary decisions

- How can we beat the lower bound?
  - Why worry? $10^{12}$ items -- 40 comparisons, very fast!

- Hashing: average case is $O(1)$ for search, independent of number of elements being searched!
  - Extra credit word tracking was a prelude to hashing
Hashing

- Store keys (strings) in specific linked list
  - hash function determines which list
  - alternative hash functions
- What makes a good hash function?
- Can collisions be avoided?
  - Birthday paradox
  - collision resolution via chaining
  - worst case? average case?

```cpp
int hash(const string & s) {
    return s[0] - 'a';
}
```
Chaining/hashing

- We want chains to be short
  - need lots of hash “buckets” and a good hash function
  - index of bucket = hash(key) % TABLE_SIZE
  - TABLE_SIZE should be a prime number

- actual performance depends on load factor
  - load factor = # keys/# buckets
  - aka average chain length (pretty close)
  - load factor, on average, is 1 for “good data”