Today

- Quiz 4
- Searching basics

Assignments

- HW4 due
- Exam 1 thurs
Finding elements in kv collections

```cpp
template<typename K, typename V>
class Collection
{
public:

    ... 

    // find the value associated with the key
    bool find(const K& key, V& val) const;

    ...
};
```

Q: Given different implementations what are some ways we can compare them?

Lots of possible ways ... for example:

- readability, lines of code, number of if-statements / subcases
- how much memory they use (space)
- how long they take (time)
- average case, best case, worst case scenarios

We'll mainly focus on worst-case **time complexity**:

- time complexity = how “long” algorithms take
What we really mean by “time”

- we focus on the number of basic operations (steps)
- we assume each step takes roughly the same amount of time
- we want to know how many “steps” needed relative to the input size

For example: “Find” a value at a particular array index

```c++
typedef pair<string, int> pair_t;

// "magically" know index to look for value
void find_val(const pair_t array[], int n, const string& k, int& v) {
    int i = magic(k); // finds i in 1 step
    v = array[i].second;
}
```

Assume:

- size of the input is $n$ (i.e., $n$ array elements)
- `array[i]` is one step (array lookup)
- accessing second is one step
- assignment is one step

As a function $T$ over input size $n > 0$, we have that $T(n) = 3$

- this is a “constant time” function (algorithm)

*Constant time functions are boring but efficient!*
Another example:

```c++
void find_val(const pair_t array[], int n, const string & k, int& v)
{
    for(int i = 0; i < n; ++i)
        if (array[i].first == k) {
            v = array[i].second;
            return;
        }
}
```

Assume:

- assignment counts as one step
- comparison counts as one step
- array access counts as one step
- accessing first and second each count as one step
- increment counts as one step
- returning is "free"

**Worst Case** = upper bound on number of steps (i.e., won't do worse)

Q: What is the “worst case” scenario for this version of find_val?

- When k is the last element in the vector!