Plan for the week

- Recurrences
  - How do we analyze the run-time of recursive algorithms?
- Setting up the Boggle assignment

Big-Oh for recursive functions

```java
class Solution {
    public int factorial(int n) {
        if (n == 0) return 1;
        return n * factorial(n - 1);
    }
}
```

- What’s the Big Oh for $n = 0, 1, 2, \ldots$?
- Express cost of algorithm for input of size $n$ as recurrence $T(n)$
  - $T(0)$ Base case!
  - $T(n) =$ if $n > 0$
- Repeated substitution to solve for a closed form answer

Solving recurrence relations

- **plug, simplify, reduce, guess, verify?**
  - $T(n) = T(n-1) + 1$
  - $T(0) = 1$
  - $T(n-1) = T$
  - $T(n) = [T(n-2) + (n+1)] + T(n-2) + 2$
  - $T(n-2) = T$
  - $T(n) = [(T(n-3) + (n+1)) + 1] + 1 = T(n-3) + 3$
  - $T(n) = T(n-k) + k$
  - find the pattern!
  - Now, let $k=n$, then $T(n) = T(0) + n = 1 + n$
  - get to base case, solve the recurrence: $O(n)$

Complexity Practice

- What is complexity of `Build`? (what does it do?)
  ```java
  public class Solution {
      public ArrayList build(int n) {
          if (0 == n) return new ArrayList(); // empty
          ArrayList list = build(n-1);
          for (int k=0; k < n; k++) {
              list.add(new Integer(n));
          }
          return list;
      }
  }
  ```
  - Write an expression for $T(n)$ and for $T(0)$, solve.
Recognizing Recurrences

- Solve once, re-use in new contexts
  - T must be explicitly identified
  - n must be some measure of size of input/parameter
    - T(n) is the time for quicksort to run on an n-element vector

\[
\begin{align*}
T(n) &= T(n/2) + O(1) & \text{binary search} & \mathcal{O}(\log n) \\
T(n) &= T(n-1) + O(1) & \text{sequential search} & \mathcal{O}(n) \\
T(n) &= 2T(n/2) + O(1) & \text{tree traversal} & \mathcal{O}(n) \\
T(n) &= 2T(n-1) + O(n) & \text{quicksort} & \mathcal{O}(n \log n) \\
T(n) &= 2T(n-1) + O(1) & \text{Towers of Hanoi} & \mathcal{O}(2^n) \\
\end{align*}
\]

- Remember the algorithm, re-derive complexity

Another recurrence

```java
int boom(int m, int x) {
    if (m == 0)
        return h(x);
    return boom(m-1, q(x)) + boom(m-1, r(x));
}
```

- Assume h(x), q(x), and r(x) are all O(1)
- Express cost of algorithm for input of size m as T(m)

Boggle Program

- Starting at board location (row, col): find a string S
  - We want to keep track of where we are in the string
  - Also track what board locations used for S search
- How do we know when we’re done?
  - Base case of recursive, backtracking call
  - Where are we in the string?
- How do we keep track of used locations?
  - Store in array list: tentatively use current one, recurse
  - If we don’t succeed, take off the last one stored!