Big Oh Again Again

- Have taken the attitude that mostly you can look things up
- Now need to be able to do your own derivations
- Extend our menu of solutions to common recurrence
- Let’s look at previously shown table

Recognizing Common Recurrences

- Below are some algorithms and recurrence relation encountered
  - 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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Recurrence Relation</th>
<th>Big Oh</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary search</td>
<td>T(n/2) + O(1)</td>
<td>O(\log n)</td>
</tr>
<tr>
<td>sequential search</td>
<td>T(n-1) + O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>tree traversal</td>
<td>T(n/2) + O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>quicksort</td>
<td>2T(n/2) + O(n)</td>
<td>O(n \log n)</td>
</tr>
<tr>
<td>selection sort</td>
<td>T(n-1) + O(n)</td>
<td>O(n^2)</td>
</tr>
</tbody>
</table>

- Remember the algorithm, re-derive complexity

Big Oh for Quickselect

- Quickselect finds the Nth Smallest item in a list
  - For example
  - \{ 13, 14, 11, 17, 15, 19, 12, 16, 18, 17\}
  - 4th smallest is 14. Program partially sorts so that it ends up in the 4th index position (3).
- Code on next slide
  - Has much in common with Quicksort
  - What are the difference?

- Recurrence Relation
  - T(0) = 1
  - T(N) = T(N/2) + N
- What is Big Oh?

Quickselect

- Partially reorders list so that kindex smallest is in proper position

```java
void quickselect(String[] list, int first, int last, int kIndex){
    int k, lastIndex = first;
    String pivot = list[first];
    for(k = first+1; k <= last; k++){
        if (list[k].compareTo(pivot) <= 0){
            lastIndex++;
            swap(list, lastIndex, k);
        }
    }
    swap(list, lastIndex, first);
    if (lastIndex == kIndex) return;
    if (kIndex < lastIndex)
        quickselect(list, first, lastIndex-1, kIndex);
    else
        quickselect(list, lastIndex+1, last, kIndex);
}
```
Solving Quickselect Big Oh

- Plug, simplify, reduce, guess, verify?

T(n) = T(n/2) + n
T(0) = 1

T(n/2) = T(n/2/2) + n/2

T(n) = [T(n/4) + n/2] + n = T(n/4) + 3n/2

T(n/4) = T(n/4/2) + n/4

T(n) = [(T(n/8) + n/4) + n/2] + n = T(n/8) + 7n/4

T(n) = (T(n/8) + n/4) + (2 - 1/2^k)n

Now, let k=log n, then T(n) = T(0) + 2n = 1 + 2n

- Get to base case, solve the recurrence: O(n)

Helpful formulae

- We always mean base 2 unless otherwise stated

  - What is log(1024)?
  - log(xy) = log(x) + log(y)
  - log(x^y) = y log(x)
  - n log(2) = n
  - 2^(log n) = n

- Sums (also, use sigma notation when possible)

  - 1 + 2 + 4 + 8 + ... + 2^k = 2^(k+1) - 1 = \sum_{i=0}^{k} 2^i
  - 1 + 2 + 3 + ... + n = n(n+1)/2 = \sum_{i=1}^{n} i
  - a + ar + ar^2 + ... + ar^{n-1} = a(r^n - 1)/(r-1) = \sum_{i=0}^{n-1} ar^i

Towers of Hanoi

```java
void hanoi(String from, String to, String via, int n) {
    if (n == 1) // base case: only one disk in pile
        System.out.println("Move disk 1 from " + from + " to " + to);
    else {
        hanoi(from, via, to, n-1); // move disks above to alternate
        System.out.println("Move disk " + n + " from " + from + " to " + to);
        hanoi(via, to, from, n-1); // move disk above to target
    }
}
```

Towers of Hanoi code
Solving Towers of Hanoi Big Oh

- **Recurrence relation:**

  \[
  T(n) = 2T(n-1) + 1 \\
  T(0) = 1 \\
  T(n-1) = 2T(n-2) + 1 + 1 = 4T(n-2) + 3 \\
  T(n-2) = T(n-2-1) + 1 \\
  T(n) = 2[2(2T(n-3) + 1) + 1] + 1 = 8T(n-3) + 7 \\
  T(n) = 2^kT(n-k) + 2^k - 1 \\
  \text{find the pattern!}
  \]

  Now, let \( k=n \), then \( T(n) = 2^nT(0) + 2^n - 1 = 2^{n+1} - 1 \)

- **Get to base case, solve the recurrence:** \( \text{O}(2^n) \)

- **Oh – Oh!**

---

Eugene (Gene) Myers

- **Lead computer scientist/software engineer at Celera Genomics (now at Berkeley)**

- "What really astounds me is the architecture of life. The system is extremely complex. It's like it was designed." ... "There's a huge intelligence there."