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 recurrences
- Let’s look at previously shown table
Recognizing Common Recurrences

❖ Below are some algorithms and recurrence relation encountered

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

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

❖ 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 differences?

- **Recurrence Relation**  
  - T(0) = 1  
  - T(N) = T(N/2) + N

- **What is Big Oh?**
Quickselect

- Partially reorders list so that \texttt{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?**

\[
T(n) = T(n/2) + n \\
T(1) = 1
\]

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

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

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

\[
T(n) = [T(n/8) + n/4] + 3n/2 = T(n/8) + 7n/4
\]

\[
T(n) = T(n/2^k) + (2 - 1/2^k)n \quad \text{find the pattern!}
\]

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

- **Get to base case, solve the recurrence: \( O(n) \)**
Helpful formulae

- We always mean base 2 unless otherwise stated

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

- Sums (also, use sigma notation when possible)

  - \( 1 + 2 + 4 + 8 + \ldots + 2^k = 2^{k+1} - 1 = \sum_{i=0}^{k} 2^i \)
  - \( 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2} = \sum_{i=1}^{n} i \)
  - \( a + ar + ar^2 + \ldots + ar^{n-1} = \frac{a(r^n - 1)}{r-1} = \sum_{i=0}^{n-1} ar^i \)
Towers of Hanoi

// Initial state for n=3

//
// A   B   C
// |
// (__) 1
// (___) 2
// (____) 3

Sample output responding to hanoi("A", "C", "B", 3);

>Move disk 1 from A to C
>Move disk 2 from A to B
>Move disk 1 from C to B
>Move disk 3 from A to C
>Move disk 1 from B to A
>Move disk 2 from B to C
>Move disk 1 from A to C
void hanoi(String from, String to, String via, int n)
// Pre: n > 0 disks in pile "from" to be moved to pile "to"
// with pile "via" available for intermediate storage. All
// piles so that disk n always above disk n+k where k > 0.
// Post: Messages generated to show how to move disks to pile "to"
// with intermediate use of all piles but only one disk moved at
// a time and at all times for all n, disk n above disk n+k where
// k > 0. (I.e., at no time is a larger disk above a smaller disk
// where smaller disks have smaller numbers than larger disks.)
{
    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
    }
}
Solving Towers of Hanoi Big Oh

- **Recurrence relation:**

\[
T(n) = 2T(n-1) + 1
\]

\[
T(0) = 1
\]

\[
T(n-1) = 2T(n-1-1) + 1 = 2T(n-2) + 1
\]

\[
T(n) = 2[2T(n-2) + 1] + 1 = 4T(n-2) + 3
\]

\[
T(n-2) = 2T(n-2-1) + 1 = 2T(n-3) + 1
\]

\[
T(n) = 4[2T(n-3) + 1] + 3 = 8T(n-3) + 7
\]

\[
T(n) = 2^kT(n-k) + 2^k - 1
\]

**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:** \( O(2^n) \)

- **Oh – Oh!**