Lecture 05: Analysis of Recursive Algorithms
Previously ...

Recursion
definition
backtracking
examples
Games in AI

Tic-Tac-Toe — PhD project @ Cambridge 1952
Games in AI

Tic-Tac-Toe — PhD project @ Cambridge 1952
Chess — Deep Blue 1997
Games in AI

Tic-Tac-Toe — PhD project @ Cambridge 1952

Chess — Deep Blue 1997

Jeopardy — Watson 2011
Games in AI

Tic-Tac-Toe — PhD project @ Cambridge 1952
Chess — Deep Blue 1997
Jeopardy — Watson 2011
Atari Games from raw pixels (Google) — 2014
Games in AI

Tic-Tac-Toe — PhD project @ Cambridge 1952

Chess — Deep Blue 1997

Jeopardy — Watson 2011

Atari Games from raw pixels (Google) — 2014

Go game — Deepmind (Google) — 2016
  search space more than a GooGol larger than Chess!!
Go Game

1,000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,00
0,000 possible configurations
Today ...

Recurrences
used to determine running time of recursive algorithms
Binary Search

Searching for k = 15
Binary Search

Searching for $k = 15$
Binary Search

Searching for $k = 15$
Binary Search

Searching for $k = 15$
Binary Search

Searching for $k = 15$
Binary Search

Searching for $k = 15$

mid  low  high
Binary Search

Searching for $k = 15$
Binary Search

Searching for k = 15
Binary Search

Searching for $k = 15$
int binary_search(int A[], int low, int high, int k) {
    // test if array is empty
    if (high < low) {
        // return value showing not found
        return NOT_FOUND;
    } else {
        // calculate midpoint index
        int mid = (low + high) / 2;

        if (A[mid] == k) {
            // key has been found
            return mid;
        } else if (A[mid] < k) {
            // key is in upper subset
            return binary_search(A, mid+1, high, k);
        } else {
            // key is in lower subset
            return binary_search(A, low, mid-1, k);
        }
    }
}
Binary Search Recurrence

Base Case: \[ T(1) = c_0 \]

Recursive Case: \[ T(n) = T(n/2) + c \]
The Iteration Method

Keep expanding the recurrence until you see a pattern, then simplify

Not trivial in all cases but it is helpful to build an intuition
The Iteration Method
The Iteration Method

\[ T(n) = T(n/2) + c \]
The Iteration Method

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

What is \( T(n/2) \)?

\[ T(n/2) = T\left(\frac{n/2}{2}\right) + c \]
The Iteration Method

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

\[ = T\left(\frac{n}{2}\right) + c + c \]

What is \( T(n/2) \)?

\[ T(n/2) = T\left(\frac{n/2}{2}\right) + c \]
The Iteration Method

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

\[ = T\left(\frac{n/2}{2}\right) + c + c \]

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

What is \( T(n/2) \)?

\[ T(n/2) = T\left(\frac{n/2}{2}\right) + c \]
The Iteration Method

\[ T(n) = T\left(\frac{n}{2}\right) + c \]

\[ = T\left(\frac{n/2}{2}\right) + c + c \]

\[ = T\left(\frac{n}{4}\right) + 2c \]

What is \(T(n/2)\)?
\[ T(n/2) = T\left(\frac{n/2}{2}\right) + c \]

What is \(T(n/4)\)?
\[ T(n/4) = T\left(\frac{n/4}{2}\right) + c \]
The Iteration Method

\[ T(n) = T(n/2) + c \]
\[ = T\left(\frac{n}{2}\right) + c + c \]
\[ = T(n/4) + 2c \]
\[ = T(n/8) + 3c \]

What is \( T(n/2) \)?
\[ T(n/2) = T\left(\frac{n}{2}\right) + c \]

What is \( T(n/4) \)?
\[ T(n/4) = T\left(\frac{n}{4}\right) + c \]
The Iteration Method

\[ T(n) = T(n/2) + c \]
\[ = T\left(\frac{n/2}{2}\right) + c + c \]
\[ = T(n/4) + 2c \]
\[ = T(n/8) + 3c \]
\[ = T(n/16) + 4c \]

What is \( T(n/2) \)?
\[ T(n/2) = T\left(\frac{n/2}{2}\right) + c \]

What is \( T(n/4) \)?
\[ T(n/4) = T\left(\frac{n/4}{2}\right) + c \]
The Iteration Method

\[ T(n) = T\left(\frac{n}{2}\right) + c \]

\[ = T\left(\frac{n/2}{2}\right) + c + c \]

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

\[ = T(n/8) + 3c \]

\[ = T(n/16) + 4c \]

\[ \vdots \]

What is \( T(n/2) \)?

\[ T(n/2) = T\left(\frac{n}{2}\right) + c \]

What is \( T(n/4) \)?

\[ T(n/4) = T\left(\frac{n}{4}\right) + c \]
The Iteration Method

\[ T(n) = T(n/2) + c \]
\[ = T\left(\frac{n/2}{2}\right) + c + c \]
\[ = T(n/4) + 2c \]
\[ = T(n/8) + 3c \]
\[ = T(n/16) + 4c \]
\[ \vdots \]
\[ = T(n/2^k) + kc \]
The Iteration Method

We already know $T(1)$ is equal to a constant $c_0$. 
The Iteration Method

We already know $T(1)$ is equal to a constant $c_0$

If we set $n/2^k = 1$ then $n = 2^k$ and $k = \log_2 n$
The Iteration Method

We already know $T(1)$ is equal to a constant $c_0$

If we set $n/2^k = 1$ then $n = 2^k$ and $k = \log_2 n$

$$T(n) = T(n/2^k) + kc$$
The Iteration Method

We already know $T(1)$ is equal to a constant $c_0$

If we set $n/2^k = 1$ then $n = 2^k$ and $k = \log_2 n$

$$T(n) = T(n/2^k) + kc$$

$$= T(1) + c \log_2 n$$
The Iteration Method

We already know $T(1)$ is equal to a constant $c_0$

If we set $n/2^k = 1$ then $n = 2^k$ and $k = \log_2 n$

$$T(n) = T(n/2^k) + kc$$

$$= T(1) + c \log_2 n$$

$$= c_0 + c \log_2 n$$
The Iteration Method

We already know $T(1)$ is equal to a constant $c_0$

If we set $n/2^k = 1$ then $n = 2^k$ and $k = \log_2 n$

$$T(n) = T(n/2^k) + kc$$

$$= T(1) + c \log_2 n = O(\log_2 n)$$

$$= c_0 + c \log_2 n$$
The Recursion Tree Method

Keep track of how much work each recursive call makes.

Total running time is the sum of the work across all layers of the tree.
The Recursion Tree Method

\[ T(1) = c_0 \]
\[ T(n) = T(n/2) + c \]
The Recursion Tree Method

\[ T(1) = c_0 \]
\[ T(n) = T(n/2) + c \]
The Recursion Tree Method

\[ T(1) = c_0 \]

\[ T(n) = T(n/2) + c \]
The Recursion Tree Method

\[ T(1) = c_0 \]
\[ T(n) = T(n/2) + c \]
The Recursion Tree Method

\[
T(1) = c_0 \\
T(n) = T(n/2) + c
\]
The Recursion Tree Method

\[ T(1) = c_0 \]
\[ T(n) = T(n/2) + c \]
The Recursion Tree Method

\[ T(1) = c_0 \]
\[ T(n) = T(n/2) + c \]

\[ = c_0 + c \log_2 n \]
The Recursion Tree Method

\[ T(1) = c_0 \]
\[ T(n) = T(n/2) + c \]

\[ = c_0 + c \log_2 n \]

\[ = O(\log_2 n) \]
Examples: use the iteration and the recursion tree methods:

mergesort

\[
\begin{align*}
T(1) &= a \\
T(n) &= 2T(n/2) + cn
\end{align*}
\]
Examples: use the iteration and the recursion tree methods:

mergesort

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

hanoi towers

\[
T(0) = 0 \\
T(n) = 2T(n - 1) + 1
\]