CSC 212
Data Structures and Abstractions
Fall 2015

Lecture 05: Recursion
Thinking Recursively

if (problem is simple) {
    Solve the problem and return solution
} else {
    Split the problem into smaller problems
    Solve smaller problems (recursively)
    Combine solutions
    Return the overall solution
}
Recursion

A function that calls itself
often used when iterative solution is not trivial

Structure

base case
solution of a trivial case

recursive call(s)
after dividing problem into smaller instances
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 \( k = 15 \)
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);
    }
}
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);
        }
    }
}
```c
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);
    }
}
```
```c
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);
    }
}
```
Recurrence Relations

Used to determine running time of recursive algorithms

Base case:
  T(1) time to solve problem of size 1
  sometimes we use T(0) instead of T(1)

Recursive case:
  T(n) time to solve problem of size n
Binary Search Recurrence

**Base Case:** \( T(1) = c_0 \)

**Recursive Case:** \( T(n) = T(n/2) + c \)
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

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

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

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

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

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

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

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

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

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

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

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

\[ = T\left(\frac{n}{8}\right) + 3c \]
The Iteration Method

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

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

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

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

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

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

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 Iteration Method

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

Is not that trivial for all recursive algorithms but it is helpful