Memoization

CS 106B

Programming Abstractions
Fall 2016
Stanford University
Computer Science Department
Tell me and I forget. Teach me and I rememoize.*

- Xun Kuang, 300 BCE

* This is almost the correct quote
Midterm

Date: Tuesday, Nov 3rd
Time: 7-9pm
Location: Braun Aud + Cemex

Hand written, four questions.

Midterm Review: Next Monday (31st of Oct)

First Practice Midterm on Wednesday

Material up until Wednesday’s class.

Open book, closed CPU
Ha ha ha, I destroyed you. Better luck next time, puny human!
Due in 2 weeks!

YEAH @ 5pm

Ready on Wednesday

Midterm Practice
Today’s Goal

1. Feel Comfort with Big O
2. Feel Comfort with Recursion
3. Understand the benefit of memoization
Real Life Problem

This doesn’t fit in instagram 😞
Bad Option 1: Crop

You got cropped out!
Bad Option 2: Resize

Weird looking castle 😞
New Algorithm: Seam Carving
Review
How do we compare algorithms?

Binary Search

Linear Search
```cpp
int find(Vector<int> & vec, int goal) {
    for (int i = 0; i < vec.size(); i++) {
        if (vec[i] == goal)
            return i;
    }
    return -1;
}
```

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

Do we really care about the 4?

Do we really care about the +2?
Ignore *everything* except the dominant growth term, including constant factors.

**Examples:**
- \(4n + 2 = \mathcal{O}(n)\)
- \(137n + 271 = \mathcal{O}(n)\)
- \(n^2 + 3n + 4 = \mathcal{O}(n^2)\)
- \(2^n + n^3 = \mathcal{O}(2^n)\)
- \(\log_2 n = \mathcal{O}(\log n)\)
• Ignore *everything* except the dominant growth term, including constant factors.

• Examples:
  
  - $4n + 2 = \mathcal{O}(n)$
  - $137n + 271 = \mathcal{O}(n)$
  - $n^2 + 3n + 4 = \mathcal{O}(n^2)$
  - $2^n + n^3 = \mathcal{O}(2^n)$
  - $\log_2 n = \mathcal{O}(\log n)$

  Keep constants in the base or exponent of a power.
• Ignore *everything* except the dominant growth term, including constant factors.

• Examples:
  
  - $4n + 2 = O(n)$
  - $137n + 271 = O(n)$
  - $n^2 + 3n + 4 = O(n^2)$
  - $2^n + n^3 = O(2^n)$
  - $\log_2 n = O(\log n)$

  *Keep constants in the base or exponent of a power.*

  *Do not keep constants in logs.*
<table>
<thead>
<tr>
<th>( \log n )</th>
<th>( n )</th>
<th>( n \log n )</th>
<th>( n^2 )</th>
<th>( 2^n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>24</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>64</td>
<td>256</td>
<td>65,536</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>160</td>
<td>1,024</td>
<td>4,294,967,296</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>384</td>
<td>4,096</td>
<td>1.84 x 10^{19}</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>896</td>
<td>16,384</td>
<td>3.40 x 10^{38}</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>2,048</td>
<td>65,536</td>
<td>1.16 x 10^{77}</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>4,608</td>
<td>262,144</td>
<td>1.34 x 10^{154}</td>
</tr>
<tr>
<td>10</td>
<td>1,024</td>
<td>10,240 (.000003s)</td>
<td>1,048,576 (.0003s)</td>
<td>1.80 x 10^{308}</td>
</tr>
<tr>
<td>30</td>
<td>1,300,000,000 (half a second)</td>
<td>390000000000 (13s)</td>
<td>1690000000000000000000 (18 years)</td>
<td>LOL</td>
</tr>
</tbody>
</table>

# of Facebook accounts
THIS IS ME NOT CARING ABOUT PERFORMANCE TUNING UNLESS IT CHANGES BIG-O
Big O Strategy

- **Non-Recursive**
  - Try and count number of executions

- **Recursive**
  - Try and count number of function calls
Pro Tip #1: Arithmetic Sum
Arithmetic Sum

• You can convince yourself that

\[ 1 + 2 + 3 + \cdots + (N - 2) + (N - 1) + N = \frac{N \times (N + 1)}{2} \]
You can convince yourself that

\[ 1 + 2 + 3 + \cdots + (N - 2) + (N - 1) + N = \frac{N \times (N + 1)}{2} \]
You can convince yourself that

\[1 + 2 + 3 + \cdots + (N - 2) + (N - 1) + N = \frac{N \times (N + 1)}{2}\]
You can convince yourself that

$$1 + 2 + 3 + \cdots + (N - 2) + (N - 1) + N = \frac{N \times (N + 1)}{2}$$
You can convince yourself that

\[ 1 + 2 + 3 + \cdots + (N - 2) + (N - 1) + N = \frac{N \times (N + 1)}{2} \]
You can convince yourself that

\[ 1 + 2 + 3 + \cdots + (N - 2) + (N - 1) + N = \frac{N \times (N + 1)}{2} \]
You can convince yourself that

\[ 1 + 2 + 3 + \cdots + (N - 2) + (N - 1) + N = \frac{N \times (N + 1)}{2} \]
You can convince yourself that

\[ 1 + 2 + 3 + \cdots + (N - 2) + (N - 1) + N = \frac{N \times (N + 1)}{2} \]
You can convince yourself that
\[
1 + 2 + 3 + \cdots + (N - 2) + (N - 1) + N = \frac{N(N + 1)}{2}
\]
You can convince yourself that

\[
1 + 2 + 3 + \cdots + (N - 2) + (N - 1) + N = \frac{N \times (N + 1)}{2}
\]

\[
= \frac{N \cdot (N + 1)}{2}
\]

\[
= \frac{1}{2}N^2 + \frac{1}{2}N
\]

\[O(N^2)\]
Pro Tip #2: Loops Multiply Big O
void friendly(Set<string>& facebookUsers) {

  int n = facebookUsers.size();
  cout << "n: " << n << endl;

  for(string user : facebookUsers){
    // addFriend is O(log n)
    addFriend(user, "Chris Gregg");
  }
}

ProTip: Big O is multiplicative in loops

What is the Big O?
End Aside
Fibonacci
int fib(int n) {
    if(n <= 1) {
        // base case
        return 1;
    } else {
        // recursive case
        return fib(n - 1) + fib(n - 2);
    }
}
int fib(int n) {
    if (n <= 1) {
        // base case
        return 1;
    } else {
        // recursive case
        return fib(n - 1) + fib(n - 2);
    }
}
```c
int fib(int n) {
    if (n <= 1) {
        // base case
        return 1;
    } else {
        // recursive case
        return fib(n - 1) + fib(n - 2);
    }
}
```
Big O?
Branching \((b)\) = decisions in worse recursive case.
Depth \((d)\) = longest chain of recursive calls.

\[ O(b^d) = O(2^n) \]
Aside
This is beyond CS106B:

$$O(1.62^n)$$
Fibonacci: Big O

\( O(1.62^n) \) technically is \( O(2^n) \)

since

\( O(1.62^n) < O(2^n) \)

We call it a “tighter” bound
End Aside
Doesn’t look good.
Call Ghost Busters!
```java
int fib(int n) {
    if (n <= 1) {
        // base case
        return 1;
    } else {
        // recursive case
        return fib(n - 1) + fib(n - 2);
    }
}
```
```c
int fib(int n) {
    if (n <= 1) {
        // base case
        return 1;
    } else {
        // recursive case
        return fib(n - 1) + fib(n - 2);
    }
}
```
```c
int fib(int n) {
    if (n <= 1) {
        // base case
        return 1;
    } else {
        // recursive case
        return fib(n - 1) + fib(n - 2);
    }
}
```

Fibonacci

- fib(2) is calculated 3 separate times when calculating fib(5)!
**Memoization**: Store previous results so that in future executions, you don’t have to recalculate them.

aka

Remember what you have already done!
Memoization

Cache:
Memoization

Cache:
Memoization

Cache:
Memoization

Cache:
Memoization

Cache:
Memoization

Cache:
Memoization

Cache:

\[ f(2) = 2 \]
Memoization

Cache:

\[ f(2) = 2 \]
Cache:

\[ f(2) = 2 \]
Memoization

Cache:

\[ f(2) = 2, \ f(3) = 3 \]
Cache:

\[ f(2) = 2, \ f(3) = 3 \]
Memoization

Cache:

\[ f(2) = 2, \ f(3) = 3 \]
Cache:

\[ f(2) = 2, \quad f(3) = 3 \]
Memoization

Cache:

\[ f(2) = 2, \, f(3) = 3 \]
Cache:

\[ f(2) = 2, \quad f(3) = 3 \]
Memoization

Cache:

\[ f(2) = 2, \ f(3) = 3, \ f(4) = 5 \]
Memoization

Cache:

\[ f(2) = 2, \quad f(3) = 3, \quad f(4) = 5 \]
Cache:

\[ f(2) = 2, \quad f(3) = 3, \quad f(4) = 5 \]
Too Fast, Too Furious
```java
int fastFibb(Map<int, int>& cache, int n) {
    // base case
    if(cache.containsKey(n)) return cache[n];
    if(n <= 1) return 1;

    // recursive case
    int result = fastFibb(cache, n-1) + fastFibb(cache, n-2);
    cache[n] = result;
    return result;
}
```

This is now a wrapper that calls `fastFibb`:

```java
int fibb(int n) {
    Map<int, int> cache;
    return fastFibb(cache, n);
}
```
```java
int fastFibb(Map<int, int>& cache, int n) {
    // base case
    if(cache.containsKey(n)) return cache[n];
    if(n <= 1) return 1;

    // recursive case
    int result = fastFibb(cache, n-1) + fastFibb(cache, n-2);
    cache[n] = result;
    return result;
}

// This is now a wrapper that calls fastFibb
int fibb(int n) {
    Map<int, int> cache;
    return fastFibb(cache, n);
}
```
Fast Fib

\(O(n)\)
Seam Carving
Seam Carving
How to Represent the Path
struct Coord {
    int row;
    int col;
};
struct Coord {
    int row;
    int col;
};

int main() {
    Coord myCoord;
    myCoord.row = 5;
    myCoord.col = 7;

    cout << myCoord.row << endl;

    Vector<Coord> path;
    return 0;
}
Lets do it!
# calls = (N) (N − 1) × … × 1
1. Feel Comfort with Big O
2. Feel Comfort with Recursion
3. Understand the benefit of memoization