01-0: Syllabus

- Office Hours
- Course Text
- Prerequisites
- Test Dates & Testing Policies
  - Try to combine tests
- Grading Policies
How to Succeed

- Come to class. Pay attention. Ask questions.
  - A question as vague as “I don’t get it” is perfectly acceptable.
  - If you’re confused, at least 4 other people are, too.

- Come by my office
  - I am very available to students.

- Start the homework assignments early

- Read the textbook. It’s one of the best textbooks around.
  - Ask questions if you don’t understand the textbook!
01-2: How to Succeed

- Start Early (clarification)
  Some of the homework assignments for this class will be very difficult. I will not expect you to be able to just sit down and complete them.
  - Start working on a problem right when it is given. Work on it until you get stuck.
  - Come by my office, and talk about the problem with me. I will not give you the solution, but I will give you a little push in the right direction.
  - Work on the problem some more, get stuck again.
  - Come by my office again, get another push.
  - Repeat, as necessary
An algorithm is a step-by-step method for solving a problem.

Each step must be well defined.

Algorithm ≠ Computer Program.

A program is an *implementation* of an algorithm.

Can have different implementations of the same algorithm

- Different Languages
- Different Coding Styles
What makes one algorithm better than another?
What makes one algorithm better than another?

- Space
  - How much memory is used
- Time
  - How long the algorithm takes to run

Often see a time/space tradeoff – we can make an algorithm run faster by giving it more space, and vice versa.
Example: Insertion Sort

\[
\begin{align*}
    i &= 2 \\
    \text{while (} i \leq N \text{) do} \\
    &\quad \text{tmp} = A[i] \\
    &\quad j = i \\
    &\quad \text{while (} j > 1 \text{) && (tmp < A[j-1]) do} \\
    &\quad \quad j-- \\
    &\quad A[j] = \text{tmp} \\
    &\quad i++
\end{align*}
\]

<table>
<thead>
<tr>
<th>Time / Operation</th>
<th># of times executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C_1)</td>
<td>1</td>
</tr>
<tr>
<td>(C_2)</td>
<td>(N)</td>
</tr>
<tr>
<td>(C_3)</td>
<td>((N-1))</td>
</tr>
<tr>
<td>(C_4)</td>
<td>((N-1))</td>
</tr>
<tr>
<td>(C_5)</td>
<td>(\sum_{j=1}^{N-1} Itr_j + 1)</td>
</tr>
<tr>
<td>(C_6)</td>
<td>(\sum_{j=1}^{N-1} Itr_j)</td>
</tr>
<tr>
<td>(C_7)</td>
<td>(\sum_{j=1}^{N-1} Itr_j)</td>
</tr>
<tr>
<td>(C_8)</td>
<td>((N-1))</td>
</tr>
<tr>
<td>(C_9)</td>
<td>((N-1))</td>
</tr>
</tbody>
</table>
Adding everything up:

\[
C_1 + C_2N + (C_3 + C_4 + C_8 + C_9)(N - 1) + C_5 \sum_{j=1}^{N-1} (Itr_j + 1) + (C_7 + C_8) \sum_{j=1}^{N-1} Itr_j
\]

\[
(\sum_{j=1}^{N-1} Itr_j)(C_5 + C_7 + C_8) + N(C_2 + C_3 + C_4 + C_5 + C_8 + C_9) + C_1 - (C_3 + C_4 + C_8 + C_9)
\]

\[
\sum_{j=1}^{N-1} Itr_j * C_{10} + N * C_{11} + C_{12}
\]
01-8: Example: Insertion Sort

\[ \sum_{j=1}^{N-1} Itr_j \cdot C_{10} + N \cdot C_{11} + C_{12} \]

- Don’t know \( Itr_1, Itr_2, \ldots \)
- What can we do?
01-9: Example: Insertion Sort

\[ \sum_{j=1}^{N-1} Itr_j \times C_{10} + N \times C_{11} + C_{12} \]

- Don’t know \( Itr_1, Itr_2, \ldots \)
- What can we do?
  - Worst case
  - Best Case
  - Average Case
Example: Insertion Sort

\[ \sum_{j=1}^{N-1} Itr_j \cdot C_{10} + N \cdot C_{11} + C_{12} \]

- Best case

\[ N \cdot C_{11} + C_{12} \]
01-11: Example: Insertion Sort

\[ \sum_{j=1}^{N-1} Itr_j \times C_{10} + N \times C_{11} + C_{12} \]

- Worst Case

\[ \sum_{j=1}^{N-1} j \times C_{10} + N \times C_{11} + C_{12} = \frac{(N)(N - 1)}{2} \times C_{10} + N \times C_{11} + C_{12} \]
**Example: Insertion Sort**

\[
\sum_{j=1}^{N-1} Itr_j * C_{10} + N * C_{11} + C_{12}
\]

- **Average Case**
  - \[
  \sum \text{Time}(i) * P(i)
  \]
  - \text{All Possible Occurrences } i

- Calculating \( P(i) \) can be difficult
- Often assume that all instances are equally likely
  - Is that always a good assumption?
• Constants & lower order terms can be ugly:

\[ \sum_{j=1}^{N-1} j*C_{10} + N*C_{11} + C_{12} = \frac{(N - 1)(N - 2)}{2} * C_{10} + N*C_{11} + C_{12} \]

• Fortunately, these constant & lower order terms don’t matter!
<table>
<thead>
<tr>
<th>$n$</th>
<th>$8n^2 - 2n - 3$</th>
<th>Time</th>
<th>$n^2$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>777</td>
<td>0.0007 sec</td>
<td>100</td>
<td>0.0001 sec</td>
</tr>
<tr>
<td>100</td>
<td>79797</td>
<td>0.0062 sec</td>
<td>10000</td>
<td>0.01 sec</td>
</tr>
<tr>
<td>1000</td>
<td>$7.998 \times 10^6$</td>
<td>7.98 sec</td>
<td>$10^6$</td>
<td>1 sec</td>
</tr>
<tr>
<td>10000</td>
<td>$7.9998 \times 10^8$</td>
<td>13 min</td>
<td>$10^8$</td>
<td>1.6 min</td>
</tr>
<tr>
<td>100000</td>
<td>$7.99998 \times 10^{10}$</td>
<td>22 hours</td>
<td>$10^{10}$</td>
<td>2 hours</td>
</tr>
<tr>
<td>1000000</td>
<td>$8 \times 10^{12}$</td>
<td>91 days</td>
<td>$10^{12}$</td>
<td>11 days</td>
</tr>
</tbody>
</table>
## 01-15: Constants

<table>
<thead>
<tr>
<th>$n$</th>
<th>$6n + 3$</th>
<th>Time</th>
<th>$99n$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>63</td>
<td>0.00006 sec</td>
<td>990</td>
<td>0.00099 sec</td>
</tr>
<tr>
<td>100</td>
<td>603</td>
<td>0.0006 sec</td>
<td>9900</td>
<td>0.0099 sec</td>
</tr>
<tr>
<td>1000</td>
<td>6003</td>
<td>0.006 sec</td>
<td>99000</td>
<td>0.099 sec</td>
</tr>
<tr>
<td>10000</td>
<td>60003</td>
<td>0.06 sec</td>
<td>990000</td>
<td>0.99 sec</td>
</tr>
<tr>
<td>100000</td>
<td>600003</td>
<td>0.6 sec</td>
<td>9900000</td>
<td>9.9 sec</td>
</tr>
<tr>
<td>1000000</td>
<td>$6 \times 10^6$</td>
<td>6 sec</td>
<td>$9.9 \times 10^7$</td>
<td>99 sec</td>
</tr>
</tbody>
</table>
01-16: Do Constants Matter?

Comparing a recursive version of Binary Search with iterative version of linear search

- Linear Search requires time $c_1 \times n$, for some $c_1$
- Binary Search requires time $c_2 \times \log(n)$, for some $c_2$

What if there is a very high overhead cost for function calls?

What if $c_2$ is 1000 times larger than $c_1$?
<table>
<thead>
<tr>
<th>Length of list</th>
<th>Time Required for Linear Search</th>
<th>Time Required for Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.001 seconds</td>
<td>0.3 seconds</td>
</tr>
<tr>
<td>100</td>
<td>0.01 seconds</td>
<td>0.66 seconds</td>
</tr>
<tr>
<td>1000</td>
<td>0.1 seconds</td>
<td>1.0 seconds</td>
</tr>
<tr>
<td>10000</td>
<td>1 second</td>
<td>1.3 seconds</td>
</tr>
<tr>
<td>100000</td>
<td>10 seconds</td>
<td>1.7 seconds</td>
</tr>
<tr>
<td>1000000</td>
<td>2 minutes</td>
<td>2.0 seconds</td>
</tr>
<tr>
<td>10000000</td>
<td>17 minutes</td>
<td>2.3 seconds</td>
</tr>
<tr>
<td>$10^{10}$</td>
<td>11 days</td>
<td>3.3 seconds</td>
</tr>
<tr>
<td>$10^{15}$</td>
<td>30 centuries</td>
<td>5.0 seconds</td>
</tr>
<tr>
<td>$10^{20}$</td>
<td>300 million years</td>
<td>6.6 seconds</td>
</tr>
</tbody>
</table>
01-18: Big-$O$ Notation

\[
O(g(n)) = \{ f(n) \mid \exists c, n_0, \text{s.t.} \quad f(n) \leq cg(n) \text{ whenever } n > n_0 \}
\]

\( f(n) \in O(g(n)) \) means:

- \( f \) is bound from above by \( g \)
- \( f \) grows no faster than \( g \)
- \( g \) is an upper bound on \( f \)
01-19: **Big-$\Omega$ Notation**

$$\Omega(g(n)) = \{ f(n) \mid \exists c, n_0, s.t. \quad cf(n) \geq g(n) \text{ whenever } n > n_0 \}$$

$f(n) \in \Omega(g(n))$ means:

- $f$ is bound from below by $g$
- $g$ grows no faster than $f$
- $g$ is a lower bound on $f$
01-20: Big-$\Theta$ Notation

\[ \Theta(g(n)) = \{ f(n) \mid \exists c_1, c_2, n_0, s.t. \\
\quad c_1 g(n) \leq f(n) \leq c_2 g(n) \]
\[ \text{whenever } n > n_0 \} \]

Alternately,

\[ \Theta(g(n)) = O(g(n)) \cap \Omega(g(n)) \]
Show:

$$3n^2 + 4n \in \Theta(n^2)$$
Show:

\[3n^2 + 4n \in \Theta(n^2)\]

\[c_1 \cdot n^2 \leq 3n^2 + 4n \leq c_2 \cdot n^2\]

True, as long as \(c_1 \leq 3 + 4/n\), \(c + 2 \geq 3 + 4/n\)

(since \(n > n_0\), we can assume \(4/n \leq 1\)
Verify:

\[ 4n^3 \notin \Theta(n^2) \]
Verify:

\[ 4n^3 \not\in \Theta(n^2) \]

\[ 4n^3 \leq c_1 n^2 \]
\[ 4n \leq c_1 \]

which is not true for any constant \( c_1 \)
01-25: **Big-$\Theta$ Notation**

- We can drop all constants and lower order terms when finding the $\Theta$ running time of a quadratic

$$\sum_{i=0}^{k} a_i n^i \in \Theta(n^k)$$
01-26: Big-$\Theta$ in Equations

- $f(n) = \Theta(g(n))$ is shorthand for $f(n) \in \Theta(g(n))$
- $n^2 + 3n + 2 = n^2 + \Theta(n)$ Means $\exists f(n) \in \Theta(n)$ such that $n^2 + 3n + 2 = n^2 + f(n)$

This can lead to some weirdness:

- $n^2 + 2n + 3 = 5n + \Theta(n^2)$
- $n^2 + 2n + 3 = 6n + O(n^4)$
01-27: Loose & Tight Bounds

- $O()$ may or may not be tight:
  - $n^2 + 2n + 3 \in O(n^2)$
  - $n^2 + 2n + 3 \in O(n^4)$

- We have a notation for tight bound ($\Theta$), and we also have a notation for a bound that is not tight:

$$o(g(n)) = \{ f(n) \mid \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq f(n) < cg(n) \text{ whenever } n \geq n_0 \}$$
01-28:  **Loose & Tight Bounds**

\[ o(g(n)) = \{ f(n) \mid \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq f(n) < cg(n) \text{ whenever } n \geq n_0 \} \]

- \(4n \in o(n^2)\)
- \(3n^2 \not\in o(n^2)\)
- \(n^2 \not\in o(n^2)\)
little-\(o\)

\[
o(g(n)) = \{ f(n) \mid \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq f(n) < cg(n) \text{ whenever } n \geq n_0 \}
\]

\[
f(n) \in o(g(n)) \implies \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0
\]
\[ \omega(g(n)) = \{ f(n) \mid \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq cg(n) < f(n) \text{ whenever } n \geq n_0 \} \]

\[ f(n) \in \omega(g(n)) \Rightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \]
True or false:

- $f(n) \in \Theta(g(n)) \rightarrow f(n) \in O(g(n))$
- $f(n) \in O(g(n)) \rightarrow g(n) \in \Omega(f(n))$
- $f(n) \in O(g(n)) \rightarrow f(n) \in o(g(n))$
- $f(n) \in o(g(n)) \rightarrow f(n) \in O(g(n))$
- $f(n) \in o(g(n)) \rightarrow f(n) \in \Theta(g(n))$
- For any two functions $f(n), g(n)$, either $f(n) \in O(g(n))$ or $f(n) \in \Omega(g(n))$
True or false:

- $f(n) \in \Theta(g(n)) \rightarrow f(n) \in O(g(n))$ True
- $f(n) \in O(g(n)) \rightarrow g(n) \in \Omega(f(n))$ True
- $f(n) \in O(g(n)) \rightarrow f(n) \in o(g(n))$ False, $n \in O(n), n \notin o(n)$
- $f(n) \in o(g(n)) \rightarrow f(n) \in O(g(n))$ True.
- $f(n) \in o(g(n)) \rightarrow f(n) \in \Theta(g(n))$ False. In fact, $f(n) \in o(g(n)) \rightarrow f(n) \notin \Theta(g(n))$
- For any two functions $f(n), g(n)$, either $f(n) \in O(g(n))$ or $f(n) \in \Omega(g(n))$. False. Consider $f(n) = n, g(n) = n^{1+\sin n}$
Summations

- We will assume you’ve seen inductive proof that 
  \[ \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \]  
  (check Appendix A in the text otherwise!)

- Can use induction to prove bounds as well. Show:

  \[ \sum_{i=1}^{n} 3^i = O(3^n) \]
\( \sum_{i=1}^{n} 3^i = O(3^n) \)

- Base Case:
  
  \( \sum_{i=1}^{1} 3^1 = 3 \leq c \times 3^1 \) as long as \( c \geq 1 \)
\[ \sum_{i=1}^{n} 3^i = O(3^n) \]

- **Recursive Case:**

\[
\begin{align*}
\sum_{i=1}^{n+1} 3^i &= \sum_{i=1}^{n} 3^i + 3^{n+1} \\
&\leq c3^n + 3^{n+1} \\
&= (1/3 + 1/c)c3^{n+1} \\
&\leq c3^{n+1}
\end{align*}
\]

As long as \( (1/3 + 1/c) \leq 1 \), or \( c \geq 3/2 \)
**01-36: Summations**

Beware! What’s wrong with this proof?

\[ \sum_{i=1}^{n} i \in O(n) \]

- **Base case:** \[ \sum_{i=1}^{1} i = O(1) \]

- **Inductive case:**

\[
\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (i + 1)
\]

\[ = O(n) + (n + 1) \]

\[ = O(n) \]
Bounding Summations

\[ \sum_{i=1}^{n} a_i \leq n \cdot a_{\text{max}} \]

for instance: \[ \sum_{i=1}^{n} i \leq n^2 \in O(n^2) \]

and

\[ \sum_{i=1}^{n} a_i \geq n \cdot a_{\text{min}} \]

for instance: \[ \sum_{i=1}^{n} i \geq 1 \cdot n \in \Omega(n) \]

(note that the bounds are not always tight!)
Splitting Summations

We can sometimes get tighter bounds by splitting the summation:

\[
\sum_{i=1}^{n} i = \sum_{i=1}^{\lfloor n/2 \rfloor} i + \sum_{i=\lceil n/2 \rceil + 1}^{n} i \\
\geq \frac{n}{2} \times 1 + \frac{n}{2} \times \frac{n}{2} \\
\geq \left(\frac{n}{2}\right)^2 \\
\in \Omega(n^2)
\]
We can split summations in more tricky ways, as well. Consider the harmonic series:

\[ H_n = \sum_{i=1}^{n} \frac{1}{n} \]

How could we split this to get a good upper bound?
We can split summations in more tricky ways, as well. Consider the harmonic series:

\[ H_n = \sum_{i=1}^{n} \frac{1}{n} \]

How could we split this to get a good upper bound?

*HINT:* The solution we are looking for is \( H_n \in \lg(n) \)
Splitting Summations

\[ H_n = \sum_{i=1}^{n} \frac{1}{i} \leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i-1} \frac{1}{2^i + j} \leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i-1} \frac{1}{2^i} \leq \sum_{i=0}^{\lfloor \lg n \rfloor} 1 \leq \lg n + 1 \]
for (i = 1; i <= n; i++)
    for (j = 1; j <= i; j++)
        sum++;
for (i = 1; i <= n; i++)
    for (j = 1; j <= i; j++)
        sum++;

\[
\sum_{i=1}^{n} i = \frac{n(n + 1)}{2} \in \Theta(n^2)
\]
for (i = 1; i < n; i = i * 2)
    for (j = 1; j < i; j++)
        sum++;

VS

for (i = 1; i < n; i = i + 1)
    for (j = 1; j < i; j = j * 2)
        sum++;
for (i = 1; i < n; i = i * 2)
    for (j = 1; j < i; j++)
        sum++;

\[ \sum_{i=1}^{\lfloor \log n \rfloor} 2^i \in \Theta(n) \]

for (i = 1; i < n; i = i + 1)
    for (j = 1; j < i; j = j * 2)
        sum++;

\[ \sum_{i=1}^{n} \log n \in \Theta(n \log n) \]
01-46: More Summations

- More information on manipulation of summations is in Appendix A
  - (pages 1147 – 1157 in the text)
- Read over Appendix A for review
- (Should also read Chapter 2 – I’m assuming that is all review for all of you. Let me know if it is not)
Summations are used to calculate running times for iterative programs.

Recursive Algorithms use Recurrence Relations.

- \( T(n) \) is a function that returns the time it takes to solve a problem of size \( n \), for a particular recursive algorithm.
- Definition of \( T(n) \) is similar to (but not the same as!) the recursive function itself.
- Usually have a base case and a recursive case.
Recurrence Relations

MergeSort(A,low,high) {
    if (low < high) {
        mid = floor ((low + high / 2))
        MergeSort(A,low,mid)
        MergeSort(A,mid+1,high)
        Merge(A,low,mid,high)
    }
}
Recurrence Relations

```plaintext
MergeSort(A, low, high) {
    if (low < high) {
        mid = floor ((low + high / 2))
        MergeSort(A, low, mid)
        MergeSort(A, mid + 1, high)
        Merge(A, low, mid, high)
    }
}
```

\[
T(0) = \Theta(1)
\]
\[
T(1) = \Theta(1)
\]
\[
T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n)
\]
How do we solve recurrence relations?

- Substitution Method
  - Guess a solution
  - Prove the guess is correct, using induction

\[
T(1) = 1\\
T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + n
\]
**Substitution Method**

- **Inductive Case**

\[
T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + n \\
\leq 2 \left( c\frac{n}{2} \log \frac{n}{2} \right) + n \\
= cn \log n - cn \log 2 + n \\
= cn \log n - cn + n \\
\leq cn \log n
\]
Substitution Method

- Base Case

\[
\begin{align*}
T(1) &= 1 \\
T(n) &\leq cn \log n \\
T(1) &\leq c \times 1 \times \log 1 \\
T(1) &\leq c \times 1 \times 0 = 0
\end{align*}
\]

Whoops! If the base case doesn’t work the inductive proof is broken! What can we do?
• Fixing the base case

Note that we only care about $n > n_0$, and for $n > 3$, recurrence does not depend upon $T(1)$ except through $T(2)$ and $T(3)$

\[
T(2) = 4 \leq 2c \log_2 2
\]
\[
T(3) = 5 \leq 3c \log_3 3
\]

(for $c > 2$)
Sometimes, the math doesn’t work out in the substitution method:

\[ T(1) = 1 \]
\[ T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1 \]

(Work on board)
Try \( T(n) \leq cn \):

\[
T(n) = T \left( \left\lfloor \frac{n}{2} \right\rfloor \right) + T \left( \left\lceil \frac{n}{2} \right\rceil \right) + 1 \\
\leq c \left\lfloor \frac{n}{2} \right\rfloor + c \left\lceil \frac{n}{2} \right\rceil + 1 \\
\leq cn + 1
\]

We did not get back \( T(n) \leq cn \) – that extra +1 term means the proof is not valid. We need to get back exactly what we started with (see invalid proof of \( \sum_{i=1}^{n} i \in O(n) \) for why this is true)
Try $T(n) \leq cn - b$:

$$
T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + 1
$$

$$
\leq c \left\lfloor \frac{n}{2} \right\rfloor - b + c \left\lceil \frac{n}{2} \right\rceil - b + 1
$$

$$
\leq cn - 2b + 1
$$

$$
\leq cn - b
$$

As long as $b \geq 1$
Substitution Method

Substitution method can verify the solution to a recurrence relation, but how can we get our original guess?

- Compare to similar problems
  - \( T(n) = 2T(\lfloor \frac{n}{2} \rfloor - 1) + 3n + 2 \) similar to
  \[ T(n) = 2T \left( \lfloor \frac{n}{2} \rfloor \right) + n \]

- Start with loose bounds, tighten them to get a tight bound
- Recursion Trees
Recursion Trees

\[ T(n) = 2T(n/2) + cn \]
\[ T(n) = T(n - 1) + cn \]
Recursion Trees

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

$$T(n) = 3T(n/4) + cn^2$$
Recursion Trees

\[ T(n) = cn^2 + T(n/4) + T(n/4) + T(n/4) \]
Recursion Trees

\[ T(n) = cn^2 + \]

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

\[ T(n/4) = 3T(n/16) + c(n/4)^2 \]
01-64: Recursion Trees

\[ T(n) = \]

\[ c n^2 \]

\[ (\frac{c}{4})^2 \]

\[ \frac{T(n)}{16} = T(n/32) + c(n/16)^2 \]
Recursion Trees

\[ cn^2 \]

\[ c(n/4)^2 \quad c(n/4)^2 \quad c(n/4)^2 \]

\[ c(n/16)^2 \quad c(n/16)^2 \quad c(n/16)^2 \quad c(n/16)^2 \quad c(n/16)^2 \quad c(n/16)^2 \quad c(n/16)^2 \quad c(n/16)^2 \]

\[ \ldots \]

\[ 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \]

\[ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \]
Recursion Trees

\[ cn^2 \]

\[ \frac{3}{4} \cdot cn^2 \]

\[ \frac{3^2}{4^2} \cdot cn \]

\[ \frac{3^i}{4^i} \cdot cn \]

\[ n^{\log_4 3} \]

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
01-67: Recursion Trees

\[
T(n) = \sum_{i=0}^{\log_4 n} \left( \frac{3^i}{4^{2i}} \right) cn^2 + \sum_{i=0}^{n^{\log_4 3}} 1
\]

\[
< \sum_{i=0}^{\log_4 n} \left( \frac{3}{4} \right)^i cn^2 + n^{\log_4 3}
\]

\[
< \sum_{i=0}^{\infty} \left( \frac{3}{4} \right)^i cn^2 + n^{\log_4 3}
\]

\[
= \frac{1}{1 - 3/4} cn^2 + n^{\log_4 3}
\]

\[
= 4cn^2 + n^{\log_4 3}
\]

\[
\in O(n^2)
\]
01-68: Recursion Trees

\[ T(n) = T(n/3) + T(2n/3) + cn \]
Recursion Trees

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

\[ \leq \log_{3/2} n \]

\[ O(n \lg n) \]
There is a small problem – this tree is actually irregular in shape!
01-71: Recursion Trees

\[ T(n) = T(n/3) + T(2n/3) + cn \]
Recursion Trees

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

\[ 2^{\frac{\log_3 n}{2}} = n^{\frac{\log_{3/2} 2}{2}} = \omega(n \lg n) \]
If we are only using recursion trees to create a guess (that we will later verify using substitution method), then we can be a little sloppy.

Show $T(n) = T(n/3) + T(2n/3) + cn \in O(n \log n)$
Consider:

\[ T(1) = 1 \]
\[ T(n) = 2T(\sqrt{n}) + \lg n \]

- The \( \sqrt{\text{ }} \) is pretty ugly – how can we make it go away?
- Rename variables!
Renaming Variables

\[ T(1) = 1 \]
\[ T(n) = 2T \left( \sqrt{n} \right) + \lg n \]

Let \( m = \lg n \), (and so \( n = 2^m \))

\[ T(2^m) = 2T \left( \sqrt{2^m} \right) + \lg 2^m \]
\[ = 2T \left( 2^{m/2} \right) + m \]
Renaming Variables

\[ T(2^m) = 2T \left( 2^{m/2} \right) + m \]

Now let \( S(m) = T(2^m) \)

\[ S(m) = T(2^m) \]
\[ = 2T \left( 2^{m/2} \right) + m \]
\[ = 2S \left( \frac{m}{2} \right) + m \]
01-77: Renaming Variables

\[ S(m) = 2S\left(\frac{m}{2}\right) + m \]
\[ \leq cm \log m \]

So:

\[ T(n) = T(2^m) \]
\[ = S(m) \]
\[ \leq cm \log m \]
\[ = c \log n \log \log n \]
01-78: Master Method

\[ T(n) = aT\left(\frac{n}{b}\right) + f(n) \]

1. if \( f(n) \in O\left(n^{\log_b a - \epsilon}\right) \) for some \( \epsilon > 0 \), then \( T(n) \in \Theta(n^{\log_b a}) \)

2. if \( f(n) \in \Theta(n^{\log_b a}) \) then \( T(n) \in \Theta(n^{\log_b a} \ast \log n) \)

3. if \( f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right) \) for some \( \epsilon > 0 \), and if \( af(n/b) \leq cf(n) \) for some \( c < 1 \) and large \( n \), then \( T(n) \in \Theta(f(n)) \)
Master Method

\[ T(n) = 9T\left(\frac{n}{3}\right) + n \]
01-80: Master Method

\[ T(n) = 9T\left(\frac{n}{3}\right) + n \]

- \( a = 9, \ b = 3, \ f(n) = n \)
- \( n^{\log_b a} = n^{\log_3 9} = n^2 \)
- \( n \in O(n^{2-\epsilon}) \)

\[ T(n) = \Theta(n^2) \]
$T(n) = T(2n/3) + 1$
01-82: Master Method

\[ T(n) = T\left(\frac{2n}{3}\right) + 1 \]

- \( a = 1, b = 3/2, f(n) = 1 \)
- \( n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1 \)
- \( 1 \in O(1) \)

\[ T(n) = \Theta(1 \ast \lg n) = \Theta(\lg n) \]
01-83: Master Method

\[ T(n) = 3T(n/4) + n \log n \]
01-84: **Master Method**

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

- \( a = 3, \ b = 4, \ f(n) = n \log n \)
- \( n^{\log_b a} = n^{\log_4 3} = n^{0.792} \)
- \( n \log n \in \Omega(n^{0.792+\epsilon}) \)
- \( 3(n/4) \log(n/4) \leq c \times n \log n \)

\( T(n) \in \Theta(n \log n) \)
01-85: Master Method

\[ T(n) = 2T(n/2) + n \log n \]
Master Method

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

- \( a = 2, b = 2, f(n) = n \log n \)
- \( n^{\log_b a} = n^{\log_2 2} = n^1 \)

Master method does not apply!

\( n^{1+\epsilon} \) grows faster than \( n \log n \) for any \( \epsilon > 0 \)

Logs grow \textit{incredibly} slowly! \( \log n \in o(n^\epsilon) \) for any \( \epsilon > 0 \)
 Proof Sketch (not all formal, see textbook for details)

We will consider the recursion tree for
\[ T(n) = aT\left(\frac{n}{b}\right) + f(n) \]
(We’ll assume that \( n \) is an exact power of \( b \), to simplify the math. See the textbook for a complete proof)
01-88: Master Method

\[
\begin{align*}
f(n) & \\
& \quad \frac{f(n)}{b} \quad \frac{f(n)}{b} \quad \frac{f(n)}{b} \\
& \quad \frac{f(n)}{b^2} \quad \frac{f(n)}{b^2} \quad \frac{f(n)}{b^2} \\
& \quad \vdots \\
& \quad O(1) \quad O(1) \quad O(1) \\
& \quad \vdots \\
& = a^{\log_b n} = n^{\log_b a}
\end{align*}
\]
Master Method

\[ T(n) = \sum_{i=0}^{\log_b n - 1} a^i f \left( \frac{n}{b^i} \right) + cn^{\log_b a} \]

- Case 1: Leaves of recursion tree dominate cost
- Case 2: Cost is evenly divided among all levels in the tree
- Case 3: Root dominates the cost

(see the textbook for the algebra)