01-0: **Syllabus**

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

01-1: **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

01-3: **What is an Algorithm?**

- An algorithm is a step-by-step method for solving a problem.
- Each step must be well defined.
- Algorithm $\neq$ Computer Program.
- A program is an *implementation* of an algorithm.
- Can have different implementations of the same algorithm
  - Different Languages

- Different Coding Styles

01-4: **Comparing Algorithms**

- What makes one algorithm better than another?

01-5: **Comparing Algorithms**

- 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

01-6: **Example: Insertion Sort**

| | Time / Operation | # of times executed |
|---|---|---|
| `i = 2` | $C_1$ | 1 |
| `while (i <= N) do` | $C_2$ | N |
| `tmp = A[i]` | $C_3$ | (N-1) |
| `j = i` | $C_4$ | (N-1) |
| `while (j > 1) && (tmp < A[j-1]) do` | $C_5$ | $\sum_{j=1}^{N-1} Itr_j + 1$ |
| `A[j] = A[j-1]` | $C_6$ | $\sum_{j=1}^{N-1} Itr_j$ |
| `j--` | $C_7$ | $\sum_{j=1}^{N-1} Itr_j$ |
| `A[j] = tmp` | $C_8$ | (N-1) |
| `i++` | $C_9$ | (N-1) |

01-7: **Example: Insertion Sort**

Adding everything up:

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

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

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

01-8: **Example: Insertion Sort**

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

- Don't know $Itr_1, Itr_2, \ldots$
- What can we do?

01-9: **Example: Insertion Sort**

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

- Don't know $Itr_1, Itr_2, \ldots$
- What can we do?
  - Worst case

- Best Case
- Average Case

01-10: **Example: Insertion Sort**

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

- Best case

$$N * C_{11} + C_{12}$$

01-11: **Example: Insertion Sort**

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

- Worst Case

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

01-12: **Example: Insertion Sort**

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

- Average Case
  - $$\sum_{\texttt{All Possible Occurrences } i} Time(i) * P(i)$$
  - Calculating $P(i)$ can be difficult
  - Often assume that all instances are equally likely
    - Is that always a good assumption?

s 01-13: **Constants**

- 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!

01-14: **Constants**

| $n$ | $8n^2 - 2n - 3$ | Time | $n^2$ | Time |
|---|---|---|---|---|
| 10 | 777 | 0.0007 sec | 100 | 0.0001 sec |
| 100 | 79797 | 0.0062 sec | 10000 | 0.01 sec |
| 1000 | $7.998 * 10^6$ | 7.98 sec | $10^6$ | 1 sec |
| 10000 | $7.9998 * 10^8$ | 13 min | $10^8$ | 1.6 min |
| 100000 | $7.99998 * 10^{10}$ | 22 hours | $10^{10}$ | 2 hours |
| 1000000 | $8 * 10^{12}$ | 91 days | $10^{12}$ | 11 days |

01-15: **Constants**

| $n$ | $6n + 3$ | Time | $99n$ | Time |
|---|---|---|---|---|
| 10 | 63 | 0.00006 sec | 990 | 0.00099 sec |
| 100 | 603 | 0.0006 sec | 9900 | 0.0099 sec |
| 1000 | 6003 | 0.006 sec | 99000 | 0.099 sec |
| 10000 | 60003 | 0.06 sec | 990000 | 0.99 sec |
| 100000 | 600003 | 0.6 sec | 9900000 | 9.9 sec |
| 1000000 | $6 * 10^6$ | 6 sec | $9.9 * 10^7$ | 99 sec |

01-16: **Do Constants Matter?**

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

- Linear Search requires time $c_1 * n$, for some $c_1$
- Binary Search requires time $c_2 * lg(n)$, 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 $c_1$?

01-17: **Constants *Do Not* Matter!**

| Length of list | Time Required for Linear Search | Time Required for Binary Search |
|---|---|---|
| 10 | 0.001 seconds | 0.3 seconds |
| 100 | 0.01 seconds | 0.66 seconds |
| 1000 | 0.1 seconds | 1.0 seconds |
| 10000 | 1 second | 1.3 seconds |
| 100000 | 10 seconds | 1.7 seconds |
| 1000000 | 2 minutes | 2.0 seconds |
| 10000000 | 17 minutes | 2.3 seconds |
| $10^{10}$ | 11 days | 3.3 seconds |
| $10^{15}$ | 30 centuries | 5.0 seconds |
| $10^{20}$ | 300 million years | 6.6 seconds |

01-18: **Big-$O$ Notation**

$$\begin{aligned} O(g(n)) &= \{f(n) \mid \exists c, n_0, s.t. \\ &\qquad f(n) \leq cg(n) \quad \text{whenever} \quad n > n_0\} \end{aligned}$$

$f(n) \in O(g(n))$ 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**

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

$f(n) \in \Omega(g(n))$ 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**

$$\begin{aligned}\Theta(g(n)) &= \{f(n) \mid \exists c_1, c_2, n_0, s.t. \\ &\quad c_1 g(n) \leq f(n) \leq c_2 g(n) \\ &\quad \text{whenever} \quad n > n_0\}\end{aligned}$$

Alternately,

$$\Theta(g(n)) = O(g(n)) \bigcap \Omega(g(n))$$

01-21: **Big-$\Theta$ Notation**
Show:

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

01-22: **Big-$\Theta$ Notation**
Show:

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

$c_1 * n^2 \leq 3n^2 + 4n \leq c_2 * n^2$

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

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

01-23: **Big-$\Theta$ Notation**
Verify:

$$4n^3 \not\in \Theta(n^2)$$

01-24: **Big-$\Theta$ Notation**
Verify:

$$4n^3 \not\in \Theta(n^2)$$

$$\begin{aligned} 4n^3 &\leq c_1 n^2 \\ 4n &\leq c_1 \end{aligned}$$

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:

$$\begin{aligned} o(g(n)) &= \{f(n) \mid \forall c > 0, \exists n_0 > 0 \ s.t. \\ & \quad 0 \leq f(n) < cg(n) \text{ whenever } n \geq n_0\} \end{aligned}$$

01-28: **Loose & Tight Bounds**

$$\begin{aligned} o(g(n)) &= \{f(n) \mid \forall c > 0, \exists n_0 > 0 \ s.t. \\ & \quad 0 \leq f(n) < cg(n) \text{ whenever } n \geq n_0\} \end{aligned}$$

- $4n \in o(n^2)$
- $3n^2 \notin o(n^2)$
- $n^2 \notin o(n^2)$

01-29: **little-$o$**

$$\begin{aligned} o(g(n)) &= \{f(n) \mid \forall c > 0, \exists n_0 > 0 \ s.t. \\ & \quad 0 \leq f(n) < cg(n) \text{ whenever } n \geq n_0\} \end{aligned}$$

$$f(n) \in o(g(n)) \Rightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

01-30: **little-$\omega$**

$$\begin{aligned} \omega(g(n)) &= \{f(n) \mid \forall c > 0, \exists n_0 > 0 \ s.t. \\ & \quad 0 \leq cg(n) < f(n) \text{ whenever } n \geq n_0\} \end{aligned}$$

$$f(n) \in \omega(g(n)) \Rightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

01-31: $O, o, \Omega, \omega, \Theta$
True or false:

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

01-32: $O, o, \Omega, \omega, \Theta$
True or false:

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

01-33: **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:

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

01-34: **Summations**

$\sum\limits_{i=1}^{n} 3^i = O(3^n)$

- Base Case:
  - $\sum\limits_{i=1}^{1} 3^1 = 3 \leq c * 3^1$ as long as $c \geq 1$

01-35: **Summations** $\sum\limits_{i=1}^{n} 3^i = O(3^n)$

- Recursive Case:

$$\begin{aligned}
\sum_{i=1}^{n+1} 3^i &= \sum_{i=1}^{n} 3^i + 3^{n+1} \\
&\leq c3^n + 3^{n+1} \\
&= (1/3 + 1/c)c3^{n+1} \\
&\leq c3^{n+1}
\end{aligned}$$

As long as $(1/3 + 1/c) \leq 1$, or $c \geq 3/2$

01-36: **Summations**

Beware! What's wrong with this proof?

$\sum\limits_{i=1}^{n} i \in O(n)$

- Base case: $\sum\limits_{i=1}^{1} i = O(1)$
- Inductive case:

$$\begin{aligned}
\sum_{i=1}^{n+1} i &= \sum_{i=1}^{n} i + (i+1) \\
&= O(n) + (n+1) \\
&= O(n)
\end{aligned}$$

01-37: **Bounding Summations**

$$\sum_{i=1}^{n} a_i \leq n * a_{max}$$

for instance: $\sum\limits_{i=1}^{n} i \leq n^2 \in O(n^2)$

and

$$\sum_{i=1}^{n} a_i \geq n * a_{min}$$

for instance: $\sum_{i=1}^{n} i \geq 1 * n \in \Omega(n)$

(note that the bounds are not always tight!)

01-38: **Splitting Summations**

We can sometimes get tighter bounds by splitting the summation:

$$\begin{aligned}\sum_{i=1}^{n} i &= \sum_{i=1}^{\lfloor n/2 \rfloor} i + \sum_{i=\lfloor n/2 \rfloor + 1}^{n} i \\ &\geq n/2 * 1 + n/2 * n/2 \\ &\geq (n/2)^2 \\ &\in \Omega(n^2)\end{aligned}$$

01-39: **Splitting Summations**

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?

01-40: **Splitting Summations**

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 $H_n \in \lg(n)$

01-41: **Splitting Summations**

$$\begin{aligned}H_n &= \sum_{i=1}^{n} \frac{1}{i} \\ &\leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i - 1} \frac{1}{2^i + j} \\ &\leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i - 1} \frac{1}{2^i} \\ &\leq \sum_{i=0}^{\lfloor \lg n \rfloor} 1 \\ &\leq \lg n + 1\end{aligned}$$

01-42: **Summations & Code**

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

01-43: **Summations & Code**

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

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \in \Theta(n^2)$$

01-44: **Summations & Code**

```
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++;
```

01-45: **Summations & Code**

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

$$\sum_{i=1}^{\lg n} 2^i \in \Theta(n)$$

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

$$\sum_{i=1}^{n} \lg n \in \Theta(n \lg n)$$

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)

01-47: **Recursive Algorithms**

- 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

01-48: **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)
   }
}
```

01-49: **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)
   }
}
```

$$
\begin{aligned}
T(0) &= \Theta(1) \\
T(1) &= \Theta(1) \\
T(n) &= T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \Theta(n)
\end{aligned}
$$

01-50: **Recurrence Relations**

- How do we solve recurrence relations?
  - Substitution Method
    - Guess a solution
    - Prove the guess is correct, using induction

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \end{aligned}$$

01-51: **Substitution Method**

- Inductive Case

$$\begin{aligned} T(n) &= 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \\ &\leq 2\left(c\frac{n}{2}\lg\frac{n}{2}\right) + n \\ &= cn\lg n - cn\lg 2 + n \\ &= cn\lg n - cn + n \\ &\leq cn\lg n \end{aligned}$$

01-52: **Substitution Method**

- Base Case

$$\begin{aligned} T(1) &= 1 \\ T(n) &\leq cn\lg n \\ T(1) &\leq c*1*\lg 1 \\ T(1) &\leq c*1*0 = 0 \end{aligned}$$

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

01-53: **Substitution Method**

- 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 $T(3)$

$$\begin{aligned} T(2) &= 4 \leq 2*c*\lg 2 \\ T(3) &= 5 \leq 3*c*\lg 3 \end{aligned}$$

(for $c > 2$) 01-54: **Substitution Method**

- Sometimes, the math doesn't work out in the substitution method:

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

(Work on board)

01-55: **Substitution Method** Try $T(n) \leq cn$:

$$\begin{aligned} T(n) &= T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1 \\ &\leq c\left\lfloor \frac{n}{2} \right\rfloor + c\left\lceil \frac{n}{2} \right\rceil + 1 \\ &\leq cn + 1 \end{aligned}$$

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)

01-56: **Substitution Method** Try $T(n) \leq cn - b$:

$$\begin{aligned} T(n) &= T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1 \\ &\leq c\left\lfloor \frac{n}{2} \right\rfloor - b + c\left\lceil \frac{n}{2} \right\rceil - b + 1 \\ &\leq cn - 2b + 1 \\ &\leq cn - b \end{aligned}$$

As long as $b \geq 1$

01-57: **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

01-58: **Recursion Trees**

$$T(n) = 2T(n/2) + cn$$

01-59: **Recursion Trees**

$$T(n) = T(n-1) + cn$$

01-60: **Recursion Trees**

$$T(n) = T(n/2) + c$$

01-61: **Recursion Trees**

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

01-62: **Recursion Trees**

$$T(n) = \quad cn^2 \quad +$$

$$T(n/4) \quad + \quad T(n/4) \quad + \quad T(n/4)$$

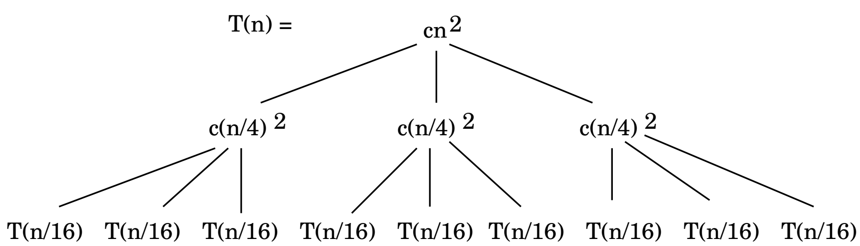
01-63: **Recursion Trees**

$$T(n) = \quad cn^2 \quad +$$

$$T(n/4) \quad + \quad T(n/4) \quad + \quad T(n/4)$$

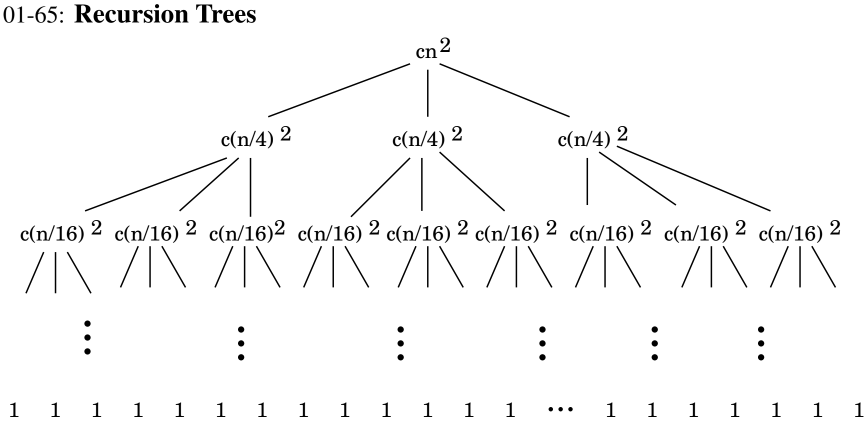
$$T(n/4) = 3T(n/16) + c(n/4)^2$$

01-64: **Recursion Trees**

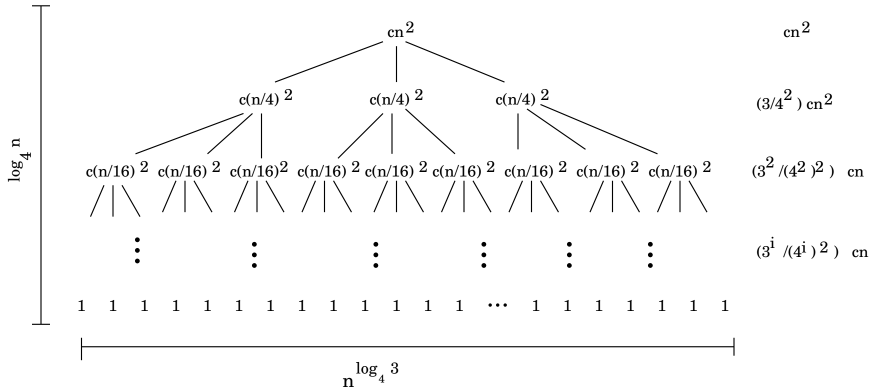



$$T(n/16) = T(n/32) + c(n/16)^2$$

01-65: **Recursion Trees**



01-66: **Recursion Trees**



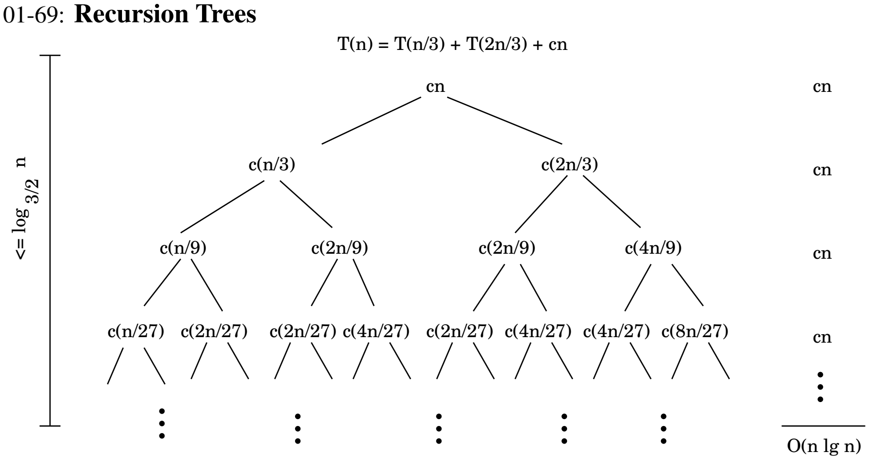

01-67: **Recursion Trees**

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n} \left(\frac{3^i}{4^{2i}}\right) cn^2 + \sum_{i=0}^{n^{\log_4 3}} 1 \\
&< \sum_{i=0}^{\log_4 n} \left(\frac{3}{4}\right)^i cn^2 + n^{\log_4 3} \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i cn^2 + n^{\log_4 3} \\
&= \frac{1}{1-3/4} cn^2 + n^{\log_4 3} \\
&= 4cn^2 + n^{\log_4 3} \\
&\in O(n^2)
\end{aligned}
$$

(now prove bound using substitution method) 01-68: **Recursion Trees**

T(n) = T(n/3) + T(2n/3) + cn

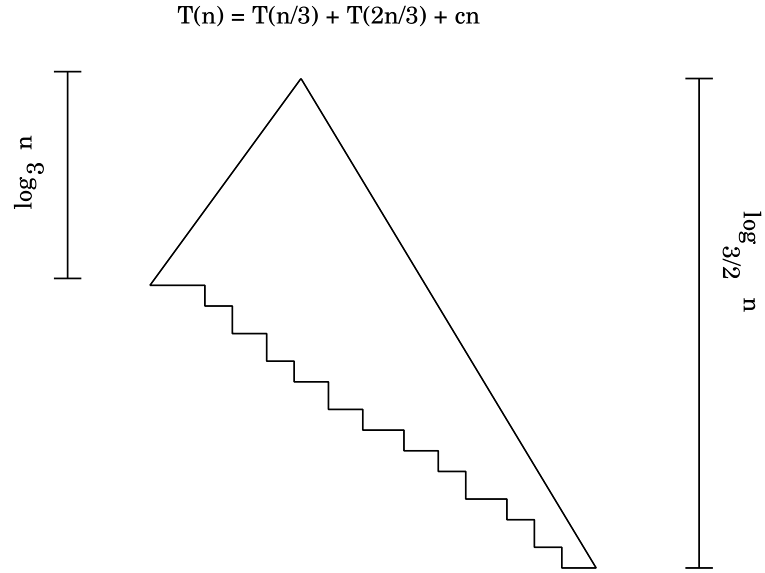
01-69: **Recursion Trees**



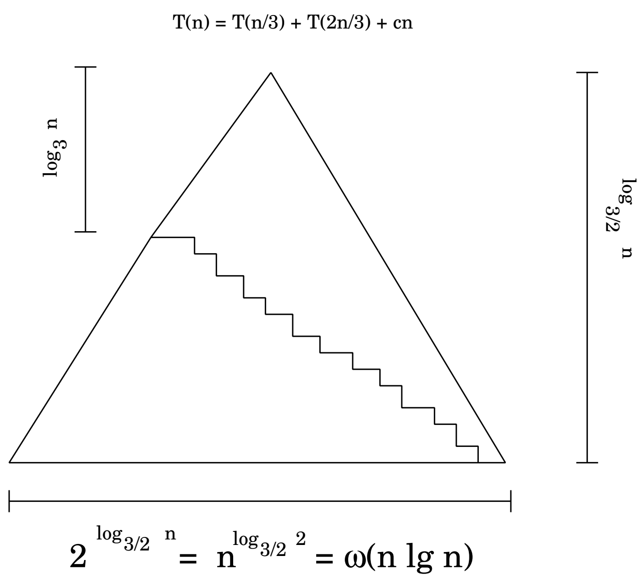

01-70: **Recursion Trees**

- There is a small problem – this tree is actually irregular in shape!

01-71: **Recursion Trees**




01-72: **Recursion Trees**




01-73: **Recursion Trees**

- 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 $T(n) = T(n/3) + T(2n/3) + cn \in O(n \lg n)$

01-74: **Renaming Variables**

- Consider:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T\left(\sqrt{n}\right) + \lg n \end{aligned}$$

- The $\sqrt{\ }$ is pretty ugly – how can we make it go away?
- Rename variables!

01-75: **Renaming Variables**

$$\begin{array}{rcl} T(1) & = & 1 \\ T(n) & = & 2T\left(\sqrt{n}\right) + \lg n \end{array}$$

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

$$\begin{array}{rcl} T(2^m) & = & 2T\left(\sqrt{2^m}\right) + \lg 2^m \\ & = & 2T\left(2^{m/2}\right) + m \end{array}$$

01-76: **Renaming Variables**

$$T(2^m) \quad = \quad 2T\left(2^{m/2}\right) + m$$

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

$$\begin{array}{rcl} S(m) & = & T(2^m) \\ & = & 2T\left(2^{m/2}\right) + m \\ & = & 2S\left(\frac{m}{2}\right) + m \end{array}$$

01-77: **Renaming Variables**

$$\begin{array}{rcl} S(m) & = & 2S\left(\frac{m}{2}\right) + m \\ & \leq & cm \lg m \end{array}$$

So:

$$\begin{array}{rcl} T(n) & = & T(2^m) \\ & = & S(m) \\ & \leq & cm \lg m \\ & = & c \lg n \lg \lg n \end{array}$$

01-78: **Master Method**

$$T(n) = aT(n/b) + f(n)$$

1. if $f(n) \in O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$
2. if $f(n) \in \Theta(n^{\log_b a})$ then $T(n) \in \Theta(n^{\log_b a} * \lg n)$
3. if $f(n) \in \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some $c < 1$ and large $n$, then $T(n) \in \Theta(f(n))$

01-79: **Master Method**

$$T(n) = 9T(n/3) + n$$

01-80: **Master Method**

$$T(n) = 9T(n/3) + n$$

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

$T(n) = \Theta(n^2)$

01-81: **Master Method**

$$T(n) = T(2n/3) + 1$$

01-82: **Master Method**

$$T(n) = T(2n/3) + 1$$

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

$T(n) = \Theta(1 * \lg n) = \Theta(\lg n)$

01-83: **Master Method**

$$T(n) = 3T(n/4) + n \lg n$$

01-84: **Master Method**

$$T(n) = 3T(n/4) + n \lg n$$

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

$T(n) \in \Theta(n \lg n)$

01-85: **Master Method**

$$T(n) = 2T(n/2) + n \lg n$$

01-86: **Master Method**

$$T(n) = 2T(n/2) + n \lg n$$

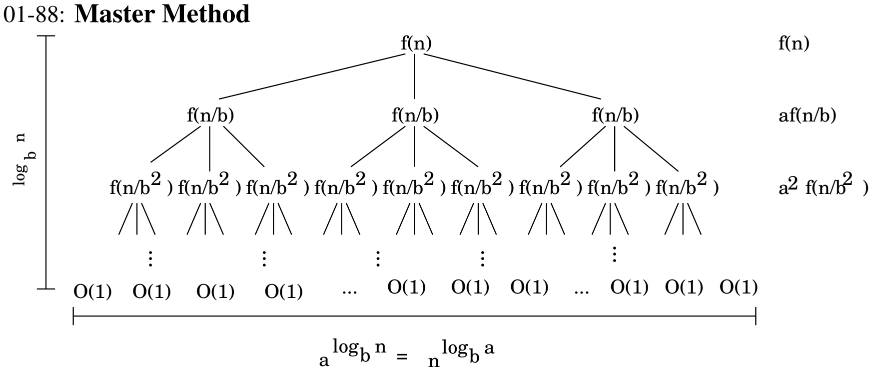
- $a = 2, b = 2, f(n) = n \lg n$
- $n^{\log_b a} = n^{\log_2 2} = n^1$

Master method does not apply!
$n^{1+\epsilon}$ grows faster than $n \lg n$ for *any* $\epsilon > 0$
Logs grow *incredibly* slowly! $\lg n \in o(n^{\epsilon})$ for any $\epsilon > 0$

01-87: **Master Method**

- Proof Sketch (not all formal, see textbook for details)
- We will consider the recursion tree for $T(n) = aT(n/b) + f(n)$
  (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**




01-89: **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 amoung all levels in the tree
- Case 3: Root dominates the cost

(see the textbook for the algebra)