CSE 3101 Design and Analysis of Algorithms
Meta Steps for Unit 0

Jeff Edmonds

This contains the **most important** concepts in this unit. You will not be able to pass this course without knowing and understanding these. The steps provided **must** be followed on all assignments and tests in this course. Do **not** believe that because you know the material, you can answer the questions in your own way. Though this material is necessary, it does not contain everything that you need. You must read the book, go to class, review the slides, and ask lots of question.

Chapter 22: Existential and Universal Quantifiers. Proofs.

People are challenged proving things. It certainly helps to have good intuition about what needs to be proved and the statements that you know are true. It also helps to be able to mechanically and meticulously to be able to follow the formal proof steps. Do not skip any.

A proof is a sequence of statement in which each is either an axiom known to be true or follows logically from previous statements. It helps to clearly state your goals, write each statement on a separate line, and indent for each line how you know it is true.

To be able to understand and prove formal statements, it is important to have intuition, but it is also important to write down what is true in a formal statement. First order logic is very important and is one of the things that students consistently get wrong.

- $\forall x\ \exists y\ x + y = 5$ is true. Let x be an arbitrary real value and let $y = 5 - x$. Then $x + y = 5$.
- $\exists y\ \forall x\ x + y = 5$ is false. Let y be an arbitrary real value and let $x = 6 - y$. Then $x + y \neq 5$.

These are the steps to follow when you prove that such a statement is true or false or when you are writing such a statement yourself. Be sure to play the correct game as to who is providing what value:

Direction: Move left to right through expression.

Exists: $(\exists y)$ means that you the prover get to chose your favorite value for y. In the proof write, "Let y be the value 5.

Forall: $(\forall x)$ means an adversary chooses the worst case object for x. In the proof write, "Let x be an arbitrary value."

Truth of Predicate: After an object has been chosen for each variable, check if the predicate is true or not.

Truth of Statement: If there is a strategy for you to win, no matter how the adversary plays, then the statement is true. If there is a strategy for the adversary to win, no matter how you play, then the statement is false.

Proving False: You prove the statement false by proving that the negation of the statement is true. Note that the exists and the foralls switch and hence roles of prover and the adversary switch.

$\forall x \exists y \forall z F(x, y, z)$: The proof/game that this is true is a follows.
Let x be an arbitrary value given to me by an Adversary.
I construct the value y_x as follows .... My choice of y_x can depend on the adversary's choice of x.
Let $z_{\langle x, y_x \rangle}$ be an arbitrary value given to me by an Adversary.
I win if $F(x, y_x, z_{\langle x, y_x \rangle})$.

Examples: Use these same game ideas to help you construct statement in first order logic.

Define $P(I)$ to be the required output of the computational problem P on inputs instance I and define $A(I)$ to be the actual output of the algorithm A on inputs instance I. We want to express the statement "Problem P is computable by some algorithm."

- $\exists A$ such that A computes P.
  This is not a good first order logic statement because the word "computes" is not defined in this language.
- $\forall I,\ P(I) = A(I)$.
  This correctly expresses the fact that algorithm A gives the correct answer for problem P on every input instance I. Hence it is a correct expression for saying "A computes P." However, in the statement "Problem P is computable by some algorithm," the variable A is what is known as a *free variable* in that it is not bound to a specific object. The statement, therefore, needs either a $\forall A$ or an $\exists A$. In contrast, the variable P is bound, in that the statement says something *about* the problem P and hence P is bound to what ever object the statement is talking about. Hence, the statement needs neither a $\forall P$ nor an $\exists P$.
- $\forall I,\ \exists A,\ P(I) = A(I)$.
  This incorrectly captures the concept because it allows there to be a different algorithm for each input instance.
- $\exists A,\ \forall I,\ P(I) = A(I)$.
  This is correct. Here are a few reasons the $\exists A$ needs to come before the $\forall I$.
  - You need one algorithm that works for every instance.
  - As a game, you must first provide the algorithm before the discussion about whether it solved problem P can even begin. Once provide, you prove its correctness by testing it on all instances.
  - Think about the brackets. The statement "Problem P is computable by some algorithm" is the same as $\exists A$ such that "A computes P." The statement "A computes P" is expressed as $\forall I,\ P(I) = A(I)$. Combine these to get the correctly bracketed expression $\exists A,\ (\forall I,\ P(I) = A(I))$.

Implication: The statement $A \Rightarrow B$ is a logic statement that is either true or false.

- It means that if A is true then B is also true.
- This could be because A causes B.
- An equivalent statement, called the "counter positive," is $\neg B \Rightarrow \neg A$, because if B is not true, then A can't be true because otherwise B would be true. Hence, the statement $A \Rightarrow B$ could be true because B being false causes A to be false.
- Or maybe C causes both A and B to be true.
- Or maybe cause and effect is not involved at all.
- $A \Rightarrow B$ formally means $\neg(A$ and $\neg B)$
  "It is not true that both A and not B are true"
- Bring the negation in gives $\neg A$ or B
  If A is false, then the statement $A \Rightarrow B$ follows automatically. Similarly if B is true, then $A \Rightarrow B$ again follows.

In the proof of $A \Rightarrow B$, one officially needs to consider two cases. In the case that A is false, the statement $A \Rightarrow B$ is trivially true. In the case that A is true, one needs to do some work to prove that B is true. To be simpler, we tend to ignore the first case and simply assume A and prove B. Generally, a proof consists of a sequence of statements all that follow from the initial assumptions. When we make the new assumption that A is true, the proof is easier to read if it indents the lines for the duration of this new assumption. Then after B is proved, this indenting "stack" is popped with the conclusion that $A \Rightarrow B$. It helps to state all of your goals and conclutions in order to give the reader the heads up.

- Goal is to prove $A \Rightarrow B$.
- Assume that A is true.

- Goal is to prove B.
- ... proof of B.
- Hence B is true.

- Hence $A \Rightarrow B$ is true.

Assuming Forall: If you assume that the truth of $S_1 : \forall x\ P(x)$, then at any point in time you know that $P(x')$ is true for your favorite value x'.

- Assume $S_1 : \forall x\ P(x)$
  - Construct your favorite x'.
  - Because S_1 is true for all x it must be true for your x'.
  - Hence $P(x')$ is true.

Assuming Exists: If you assume that the truth of $S_2 : \exists y\ Q(y)$, then at any point in time you can be given such a value of y.

- Assume $S_2 : \exists y\ Q(y)$.
  - Let y' be the value of y stated to exists by S_2.
  - Hence $Q(y')$ is true.

Assuming Forall Exists: If you assume that the truth of $S_3 : \forall x\ \exists y\ R(x, y)$, then for each x' you can be given a $y_{x'}$ for which $R(x', y_{x'})$ is true. In fact, S_3 being true gives you a function from values x' of x to values $y_{x'}$ of y.

- Assume $S_3 : \forall x\ \exists y\ R(x, y)$.
  - Construct your favorite x'.
  - Because S_3 is true for all x it must be true for your x'.
  - Hence $\exists y\ R(x', y)$ is true.
  - Let $y_{x'}$ be the value of y stated to exists.
  - Hence $Q(x', y_{x'})$ is true.

Example:

- Our goal is to prove $[\exists y\ \forall x\ R(x, y)] \Rightarrow [\forall x\ \exists y\ R(x, y)]$.
- Assume that $S_1 : \exists y\ \forall x\ R(x, y)$.
  - Our goal is to prove $S_2 : \forall x\ \exists y\ R(x, y)$.
  - Following the first order game, let x' be an arbitrary value.
  - It is then my job to construct a y'.
    - Let y' be the value of y stated to exists by S_1.
    - Hence $S_3 : \forall x\ R(x, y')$ is true.
  - To complete the first order game proof of S_2, I must prove that $R(x', y')$ is true.
    - Because S_3 is true for all x it must be true for our x'.
    - Hence $R(x', y')$ is true.
  - Hence by our game $S_2 : \forall x\ \exists y\ R(x, y)$ is true.
- Hence $[\exists y\ \forall x\ R(x, y)] \Rightarrow [\forall x\ \exists y\ R(x, y)]$ is also true.

Subsets: Prove $S_1 \subseteq S_2$ by proving $\forall x,\ [x \in S_1 \Rightarrow x \in S_2]$.

Chapter 23: Time Complexity

There is one tricky thing about time complexity that people get wrong.

algorithm *Slow*(N)

⟨*pre−cond*⟩: N is an integer.

⟨*post−cond*⟩: It prints "Hi" N times.

```
begin
    loop i = 1...N
        Print "Hi"
    end loop
end algorithm
```

Though it is true that the running time of this algorithm on input instance N is $\Theta(N)$, this is not the *time complexity* of the algorithm. The time complexity of an algorithm is the running time of the algorithm as a function of the *size* of the input. Said another way, it measures how hard the problem is to solve as a function of how hard it is for me to hand you an instance. Measuring time as N is fine. But the size of the instance N is the number $n = \log_2 N$ of bits to represent the input. Hence, the time complexity of this algorithm is $\Theta(N) = \Theta(2^n)$, which is exponential. If you don't believe me, note that I could give you a 100 bit (digit) number N is a few seconds and dare you to trace out the algorithm.

Chapter 24: Logarithms and Exponentials

Logarithms $\log_2(n)$ and exponentials 2^n arise often when analyzing algorithms.

Uses: These are some of the places that you will see them.

Divide Logarithmic Number of Times: Many algorithms repeatedly cut the input instance in half. A classic example is binary search (Section **??**). If you take something of size n and you cut it in half; then you cut one of these halves in half; and one of these in half; and so on. Even for a very large initial object, it does not take very long until you get a piece of size below 1. This number is denoted by $\log_2(n)$. Here the base 2 is because you are cutting them in half. If you were to cut them into thirds, then the number of times to cut is denoted by $\log_3(n)$.

A Logarithmic Number of Digits: Logarithms are also useful because writing down a given integer value n requires $\lceil \log_{10}(n+1) \rceil$ decimal digits. For example, suppose that $n = 1,000,000 = 10^6$. You would have to divide this number by 10 six times to get to 1. Hence, by our previous definition, $\log_{10}(n) = 6$. This, however, is the number of zeros, not the number of digits. We forgot the leading digit 1. The formula $\lceil \log_{10}(n+1) \rceil = 7$ does the trick. For the value $n = 6,372,845$, the number of digits is given by $\log_{10}(6,372,846) = 6.804333$, rounded up is 7. Being in computer science, we store our values using bits. Similar arguments give that $\lceil \log_2(n+1) \rceil$ is the number of bits needed.

Height and Size of Binary Tree: A complete balanced binary tree of height h has 2^h leaves and $n = 2^{h+1} - 1$ nodes. Conversely, if it has n nodes then its height is $h \approx \log_2 n$.

Exponential Search: Suppose a solution to your problem is represented by n digits. There are 10^n such strings of n digits. Doing a blind search through them all would take too much time.

Rules: There are lots of rules about logs and exponentials that one might learn. Personally, I like to minimize it to the following.

$b^n = (\overbrace{b \times b \times b \times \ldots \times b}^{n})$: This is the definition of exponentiation. b^n is n b's multiplied together.

$b^n \times b^m = b^{n+m}$: This is simply by counting the number of b's being multiplied.

$$(\overbrace{b \times b \times b \times \ldots \times b}^{n}) \times (\overbrace{b \times b \times b \times \ldots \times b}^{m}) = \overbrace{b \times b \times b \times \ldots \times b}^{n+m}.$$

$b^0 = 1$: One might guess that zero b's multiplied together is zero, but it needs to be one. One argument for this is as follows. $b^n = b^{0+n} = b^0 \times b^n$. For this to be true, b^0 must be one.

$b^{\frac{1}{2}} = \sqrt{n}$: By definition, $\sqrt{n}$ is the positive number which when multiplied by itself gives n. $b^{\frac{1}{2}}$ meets this definition because $b^{\frac{1}{2}} \times b^{\frac{1}{2}} = b^{\frac{1}{2}+\frac{1}{2}} = b^1 == b$.

$b^{-n} = \frac{1}{b^n}$: The fact that this needs to be true can be argued in a similar way. $1 = b^{n+(-n)} = b^n \times b^{-n}$. For this to be true, b^{-n} must be $\frac{1}{b^n}$.

$(b^n)^m = b^{n \times m}$: Again we count the number of b's.

$$\overbrace{(\overbrace{b \times b \times b \times \ldots \times b}^{n}) \times (\overbrace{b \times b \times b \times \ldots \times b}^{n}) \times \ldots \times (\overbrace{b \times b \times b \times \ldots \times b}^{n})}^{m} = \overbrace{b \times b \times b \times \ldots \times b}^{n \times m}.$$

If $x = \log_b(n)$ then $n = b^x$: This is the definition of logarithms.

$\log_b(1) = 0$: This follows from $b^0 = 1$.

$\log_b(b^x) = x$ and $b^{\log_b(n)} = n$: Substituting $n = b^x$ into $x = \log_b(n)$ gives the first and substituting $x = \log_b(n)$ into $n = b^x$ gives the second.

$\log_b(n \times m) = \log_b(n) + \log_b(m)$: The number of digits to write down the product of two integers is the number to write down each of them separately (modulo rounding errors). We prove it by applying the definition of logarithms and the above rules. $b^{\log_b(n \times m)} = n \times m = b^{\log_b(n)} \times b^{\log_b(m)} = b^{\log_b(n)+\log_b(m)}$. It follows that $\log_b(n \times m) = \log_b(n) + \log_b(m)$.

$\log_b(n^d) = d \times \log_b(n)$: This is an extension of the above rule.

$\log_b(n) - \log_b(m) = \log_b(n) + \log_b(\frac{1}{m}) = \log_b(\frac{n}{m})$: This is another extension of the above rule.

$d^{c \log_2(n)} = n^{c \log_2(d)}$: This rule states that you can move things between the base to the exponent as long as you add or remove a log. The proof is as follows. $d^{c \log_2(n)} = \left(2^{\log_2(d)}\right)^{c \log_2(n)} = 2^{\log_2(d) \times c \log_2(n)} = 2^{\log_2(n) \times c \log_2(d)} = \left(2^{\log_2(n)}\right)^{c \log_2(d)} = n^{c \log_2(d)}$.

$\log_2(n) = 3.32.. \times \log_{10}(n)$: The number of bits needed to express the integer n is 3.32.. times the number of decimal digits needed. This can be seen as follows. Suppose $x = \log_2 n$. Then $n = 2^x$, giving $\log_{10} n = \log_{10}(2^x) = x \cdot \log_{10} 2$. Finally, $x = \frac{1}{\log_{10} 2} \log_{10}(n) = 3.32.. \log_{10} n$.

Which Base: We will write $\Theta(\log(n))$ with out giving an explicit base. A high school student might use base 10 as the default, a scientist base $e = 2.718..$, and computer scientists base 2. My philosophy is that I exclude the base when it does not matter. As seen above, $\log_{10}(n)$, $\log_2(n)$, and $\log_e(n)$, differ only by a multiplicative constant. In general, we ignore multiplicative constants, and hence the base used is irrelevant. I only include the base when the base matters. For example, 2^n and 10^n differ by much more than a multiplicative constant.

The Ratio $\frac{\log a}{\log b}$: When computing the ratio between two logarithms, the base used does not matter because changing the base will introduce the same constant both on the top and the bottom, which will cancel. Hence, when computing such a ratio, you can choose whichever base makes the calculation the easiest. For example, to compute $\frac{\log 16}{\log 8}$, the obvious base to use is 2, because $\frac{\log_2 16}{\log_2 8} = \frac{4}{3}$. On the other hand, to compute $\frac{\log 9}{\log 27}$, the obvious base to use is 3, because $\frac{\log_3 9}{\log_3 27} = \frac{2}{3}$.

Exercise 0.1 *(See solution in Section ??)* *Simplify the following exponentials:* $a^3 \times a^5$, $3^a \times 5^a$, $3^a + 5^a$, $2^{6log_4 n+7}$, $n^{\frac{3}{log_2 n}}$.

Chapter 25: Asymptotic Growth

Classes of Growth Rates: It is important to be able to classify functions $f(n) = c \cdot b^{an} \cdot n^d \cdot log^e(n)$ based on how quickly they grow.

c	b^a	d	e	Class	Θ	Examples
>0	>1	any	any	**Exponentials:**	$2^{\Theta(n)}$	2^n, $\frac{3^{0.001n}}{n^{100}}$
	$=1$	>0	any	**Polynomial:**	$n^{\Theta(1)}$	n^4, $\frac{5n^{0.0001}}{\log^{100}(n)}$
		$=2$	any	**- Quadratics:**	$\Theta(n^2)$	$5n^2$, $2n^2+7n+8$
		$=1$	$=1$	**- Sorting Time:**	$\Theta(n\log n)$	$5n\log n+3n$
		$=1$	$=0$	**- Linear:**	$\Theta(n)$	$5n+3$
		$=0$	>0	**Poly-Logs:**	$\log^{\Theta(1)}(n)$	$5\log^3(n)$
			$=1$	**- Logarithms:**	$\Theta(\log(n))$	$5\log(n)$
			$=0$	**Constants:**	$\Theta(1)$	5, $5+sin(n)$
		<0	any	**Poly-Decr:**	$\frac{1}{n^{\Theta(1)}}$	$\frac{1}{n^4}$, $\frac{5\log^{100}(n)}{n^{0.0001}}$
	<1	any	any	**Exp-Decr:**	$\frac{1}{2^{\Theta(1)}}$	$\frac{1}{2^n}$, $\frac{n^{100}}{3^{0.001n}}$

Asymptotic Notation: When we want to bound the growth of a function while ignoring multiplicative constants, we use the following notation.

Greek Letter	Standard Notation	My Notation	Meaning
Theta	$f(n)=\Theta(g(n))$	$f(n)\in\Theta(g(n))$	$f(n)\approx c\cdot g(n)$
BigOh	$f(n)=\mathcal{O}(g(n))$	$f(n)\leq\mathcal{O}(g(n))$	$f(n)\leq c\cdot g(n)$
Omega	$f(n)=\Omega(g(n))$	$f(n)\geq\Omega(g(n))$	$f(n)\geq c\cdot g(n)$

Bounded Between: A function like $f(n)=8+\sin(n)$, continually changes between 7 and 9 and $f(n)=8+\frac{1}{n}$ continually changes approaching 8. However, if we don't care whether it is 7, 9, or 8.829, why should we care if it is changing between them? Hence, both of these functions are included in $\Theta(1)$. On the other hand, the function $f(n)=\frac{1}{n}$ is not included, because the only constant that it is bounded below by is zero and the zero function is not included.

Chapter 26: Adding-Made-Easy Approximations

Consider $f(n)=\Theta(b^{an}\cdot n^d\cdot\log^e n)$:

b^a	d	e	Type of Sum	$\sum_{i=1}^n f(i)$	Examples	
>1	any	any	**Geometric Inc.:** **- Dominated by last term**	$\Theta(f(n))$	$\sum_{i=0}^n 2^{2^i}$	$\approx 1\cdot 2^{2^n}$
					$\sum_{i=0}^n b^i$	$=\Theta(b^n)$
					$\sum_{i=0}^n 2^i$	$=\Theta(2^n)$
$=1$	>-1	any	**Arithmetic Like:** **- Half of Terms approximately equal**	$\Theta(n\cdot f(n))$	$\sum_{i=1}^n i^d$	$=\Theta(n\cdot n^d)=\Theta(n^{d+1})$
					$\sum_{i=1}^n i^2$	$=\Theta(n\cdot n^2)=\Theta(n^3)$
					$\sum_{i=1}^n i$	$=\Theta(n\cdot n)=\Theta(n^2)$
					$\sum_{i=1}^n 1$	$=\Theta(n\cdot 1)=\Theta(n)$
					$\sum_{i=1}^n \frac{1}{i^{0.99}}$	$=\Theta(n\cdot\frac{1}{n^{0.99}})=\Theta(n^{0.01})$
	-1	$=0$	**Harmonic:**	$\Theta(\ln n)$	$\sum_{i=1}^n \frac{1}{i}$	$=\log_e(n)+\Theta(1)$
	<-1	any	**Bounded Tail:** **- Dominated by first term**	$\Theta(1)$	$\sum_{i=1}^n \frac{1}{i^{1.001}}$	$=\Theta(1)$
					$\sum_{i=1}^n \frac{1}{i^2}$	$=\Theta(1)$
<1	any	any			$\sum_{i=1}^n (\frac{1}{2})^i$	$=\Theta(1)$
					$\sum_{i=0}^n b^{-i}$	$=\Theta(1)$

Chapter 27: Recurrence Relations

Consider $T(n)=a\cdot T\left(\frac{n}{b}\right)+f(n)$, where $f(n)=\Theta(n^c)$

$\frac{\log a}{\log b}$ vs c	Dominated By	$T(n)$	Example $\left(\frac{\log_3 9}{\log_3 3}=2\right)$	Solution
$<$	Top Level	$\Theta(f(n))$	$T(n)=9\cdot T\left(\frac{n}{3}\right)+n^4$	$\Theta(n^4)$
$=$	All Levels	$\Theta(f(n)\log n)$	$T(n)=9\cdot T\left(\frac{n}{3}\right)+n^2$	$\Theta(n^2\log n)$
$>$	Base Cases	$\Theta\left(n^{\frac{\log a}{\log b}}\right)$	$T(n)=9\cdot T\left(\frac{n}{3}\right)$	$\Theta(n^2)$

Consider $T(n) = a \cdot T(n-b) + f(n)$, where $f(n) = \Theta(n^c)$.

a	$f(n)$	Dominated By	$T(n)$	Example	Solution
> 1	any	Base Cases	$\Theta(a^{\frac{n}{b}})$	$T(n) = 9 \cdot T(n-3) + n^4$	$\Theta(9^{\frac{n}{3}})$
$= 1$	≥ 1	All Levels	$\Theta(n \cdot f(n))$	$T(n) = T(n-3) + n^4$	$\Theta(n^5)$
	$= 0$	Base Cases	$\Theta(1)$	$T(n) = T(n-3)$	$\Theta(1)$