Lecture 1 & lecture 2 Inductions
*Make sure the "chain of implications" is connected **everywhere. Note: 会分 case 讨论，比如分奇偶**

Simple induction	Complete Induction	Structural Induction (recursively defined set)
1. Define the predicate P(n) 2. Base Case: show that P(1) is True 3. Induction Step: Assume P(k) is true (I.H.), show that P(k+1) is true. EX. Using 6 cents and 11 cents can make any amount greater than 60 cents. 要点: 至少一个 11 cents 和没有 11 cents 这两个情况你都能 凑出一个 1	1. Define the predicate P(n) 2. Base Case: show that P (1) are True 3. Induction Step: Assume P(1), p(2)....p(k) are true (I.H.), show that P(k+1) is true. EX. Prove "Prime or Product of Primes" Base case: n=2, 2 is already a prime. Induction Step: Assume P(2) $\wedge$ P(3) $\wedge$ ... $\wedge$ P(n), all numbers from 2 to n can be written as a product of primes. (I.H.). Show P(n+1), can be written as a product of primes ● Case 1: n+1 is prime, then n+1 is already a product of primes, done ● Case 2: n+1 is composite (not prime), then n+1 can be written as a*b, where $2 <= a, b <= n$, According to I.H., each of a and b can be written as a product of primes. So n = a x b can be written as a product of primes.	1.Base element 2.recursive rules that generates new elements of the set from the existing element of the set (the smallest set which contains nothing else) Suppose that S is a recursively-defined set and P is some predicate ● Base case: If P is true for each base element of S ● Induction Step: under the assumption that P(e) is true for element e of S by I.H., we find that each recursive rule generates an element that satisfies P. Then P is true for all elements of S. **We must do the induction step for every recursive rule!** EX. Empty string and 1 are in S, if w is a string in S, then so are w00 and w01. Prove that S does not contain two consecutive 1s. Base Case: empty string: no consecutive 1's, "1": no consecutive 1's, check Induction Step: check all the recursive rules: Rule 1: w → w00: show P(w) $\Rightarrow$ P(w00) Rule 2: w → w01: show P(w) $\Rightarrow$ P(w01) (这里的 w 就是 empty string 或 1，100 和 101 都没有 consecutive 1s)

Lecture 3 big oh, big theta, big omega and Iterative runtime (重点就是找 c 和 n0, n0 the breakpoint 之后的 runtime，n0 一般选 1 都 work)
*(slowest) $1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < 2^n < n^n$ (fastest)

Big oh (upper bound-no faster than) Over-Estimate	Big omega (lower bound-no slower than) Under-Estimate	Big theta (tight bound-no faster and slower)
f(n) is Big-O of g(n), f(n) $\in$ O(g(n)), iff $\exists c \in$ R+, $\exists n_0 \in$ N, such that $\forall n \geq n_0$, f(n) $\leq$ c g(n) **Tricks：remove the negative term or multiply a positive term(一定要确保他是正的, 如果有可能是正的就不能 remove，一定要看好 n0 的值)** EX. Proof that 100n + 10000 is in O(n²) Choose $n_0=1$, c = 10100, then for all n >= n_0, $100n + 10000 <= 100n^2 + 10000n^2$ (**because $n \geq 1$**) $= 10100n^2 = cn^2$ Therefore, by definition of big-Oh, 100n + 10000 is in O(n²)	Function f(n)= Ω(g(n)) iff $\exists c \in$ R+, $\exists n_0 \in$ N, such that $\forall n \geq n_0$, cg(n) $\leq$ f(n) **tricks: 1.remove a positive term or multiply a negative term(一定要确保他的正负性，需要 n0 多大就选多大) 2. 把 higher order term 拆开去减那个负 term 后一起删掉** **EX.** Prove that $2n^3 - 7n + 1 = \Omega(n^3)$ $2n^3 - 7n + 1 \rightarrow n^3 + n^3 + (n^3 - 7n) + 1$(Pick $n_0 = 3$ we want $n^3 - 7n > 0$) $\rightarrow n^3 + 1$(remove a positive term) $\rightarrow n^3 \rightarrow$ pick c = 1	Function f(n) = Θ(g(n)) iff f(n) $\in$ O(g(n)) and f(n)= Ω(g(n)) Proof: big oh 和 big omega 都 prove 一遍

Worst case analysis of iterative runtime:把每层 loop 运行的次数用 summation$\sum$表示出来，一般一个 loop 就是 0 到 n-1

Lecture 4 recursion (find the closed form thru repeated substitution, develop a recurrence)
我们是通过带有 F(n)的 recursive function，用 repeated substitution 得到一个 closed form，也就是通式，知道了 n 就直接知道了 runtime，通过 closed form 去找这个 recursive function 的 runtime，而带有 T(n)的式子就是 recurrence 了，是以 recursive 的形式来表示的 runtime

repeated substitution	Prove the closed form using induction
substitute 几次，substitution 的次数就是 k，substitute 的时候**常数项尽量不要拆开**方便找规律，找到一个 pattern 以后以 k 和 n 的形式总结出来 guessed closed form，**for each base case,** 把 function 用 base case 代进去替换掉，得出 k，plug k back into the formula, prove the closed form is equivalent to the recurrence by induction	EX. prove T1(n) = 2T(n-1) + 1 (the recurrence) is equivalent to T0(n) = 2^n-1(closed form) Predicate P(n): T1(n) = T0(n) Base case: T1(1) = 1 = 2^1-1=1(把 base case 分别带入 recurrence 和 closed form 中) Induction step：suppose $n > 1$ and that T1(k)=T0(k), prove T1(k+1) = T0(k+1)

Develop a recurrence:1. Find the base case(s) that can be evaluated directly 2. break the large problem into small problems, so that you can define f(n) in terms of f(m) for some m < n.(e.g. f(n): The number of 2-element subsets of n elements. 分成 the subsets that contain e_n and the subsets that do NOT contain e_n, 有 e_n 的那组，他和除了他自己的都可以组，所以有 n-1 组，那没有 e_n 的那组，就少了一个 element，就是有 f(n-1)组啦！所以最后 f(n)=n-1+f(n-1))

Lecture 5 Recursion and Master theorem (Prove the runtime of recursive functions)

	Master Theorem: （找的是 asymptotic bound）
如果直接给你一个 **recursive function** 怎么看他的 runtime？constant 就写 constant，F(n) 就写 T(n)，F(n-1)就写 T(n-1)，和 base case 结合起来，**写成花括号的形式**，，然后再继续用 repeated substitution 等接下去一系列的操作求出这个 recursive function 的 runtime。 求 recurrence 的 runtime 的三种方法: 1. find the recurrence and use repeated substitution to find the closed form, prove the theta bound. 2. 直接猜一个 runtime，用 induction 证（例题在下面） 3.Master theorem	Let T(n) be defined by the reccurence T(n) = aT(n/b) + f(n), for some constants a $\geq$1，b $>$ 1 and k $\geq$1，then we can conclude the following about the asymptotic complexity of T(n): 注意一定要满足形式 (1) If k = $\log_b a$, then T(n) = O(n^k log n). (2) If k $< \log_b a$, then T(n) = O($n^{\log_b a}$). (3) If k $> \log_b a$, then T(n) = O(n^k). When master theorem does not directly apply,把小的合到大的那个里在用

Lecture 6 Divide and Conquer (写满足这种形式 T(n) = aT(n/b) + f(n)的 function)
一定要注意的是最终的结果是不是可以直接来源于左右 sub-problems，还是要 cross the middle point!一般最后的 n 就是用来 check middle point 的

Lecture 7 & Lecture 8 Recursive program correctness and Iterative program correctness

Recursive Program Correctness (for each program path)	Iterative Program Correctness (partial correctness and termination)
○ if there is no recursive call, analyze the code directly (like base cases) ○ if there are recursive calls 1.show that the precondition holds when making each recursive call 2. Show that the recursive call occurs on "smaller" values than the original call. (So it will terminate eventually) Note: sometimes measure of the size 是两个 parameter 的 sum ! 3. You can then assume that the recursive call satisfies the post-condition (by Induction Hypothesis) 4. Show that the post-condition of the function is satisfied based on the assumption **5.conclusion**	**Partial Correctness: (induction step 时可能需要分奇偶讨论)** **loop invariant: relationship of parameters that's not going to change in every iteration，和 loop guard 两部分 (track the code for several iteration to find the invariant)** 1. Base case: Argue that the loop invariant is true when the loop is reached（还没进到 loop 之前） 2. Induction Step: assume that the invariant and guard are true at the end of an arbitrary iteration **i0** (by I.H.), show that the invariant remains true after one iteration **i1**(invariant 和 loop guard 两个部分分别证，**表示出 variable 在两个 iteration 之间的变化**，比如 i1=i0+1, sum1=sum0+A[i0]) 3. Check post-condition: Argue that the invariant and the negation of the loop guard together let us conclude the program's post-condition. **Termination:(loop variant)** **Show that the loop variant is a natural number that's decreasing on every iteration** If the loop variant decreases on each iteration yet cannot drop below 0, then we can conclude that at some point the loop must terminate. (sometimes it's 一个 variable 加/减另一个 variable) 4.**conclusion**

Lecture 9 Regular Languages and Regular Expressions (基本只是 terminologies，description to regex, regex to description, simplify regex)

1. Alphabet: a finite set of symbols 2. A string w over alphabet $\sum$ is a finite sequence of symbols from $\sum$, 由 alphabet 里的 elm 组成的一个 string 3. Empty string "" which we denote with ε, it's a string over any alphabet. 4. Length of string: number of characters in w, $|\varepsilon| = 0$, $\sum^n$ is set of strings over $\sum$ of length n, $\sum^*$ is set of all strings over $\sum$	5. A language L is only a subset of $\sum^*$, L $\subseteq \sum^*$ 6. Operations on Languages: Given two languages L, M $\subseteq \sum^*$, three operations can be used to generate new languages. - Union, L $\cup$ M - Concatenation, LM: L 里的每个 elm 都和 M 里的每个 elm 相连 - L*: all strings that can be formed by concatenating 0 or more strings from L.

7. A regular expression (regex) is a string representation of a regular language. A regex "matches" a set of strings (the represented regular language).

Definition of regular language (a set of strings)	Definition of regular expression (for a regex r, L(r) is the language matched by r)
1. ∅, the empty set, is a regular language 2. {ε}, the language consisting of only the empty string, is a regular language 3. For any symbol a $\in \sum$, {a} is a regular language. 4. If L, M are regular languages, then so are L $\cup$ M, LM, and L*(recursive rule) EX. Prove that language L = {a, aa} is a regular language. Proof: {a} is regular by definition So {aa} = {a} {a} is regular (concatenation rule) So {a, aa} = {a} U {aa} is regular (union rule)	1. ∅ is a regex, with L(∅)string with the single character ∅ = ∅ empty set. (matches no string) 2. eps is a regex, with L(epsilon) = {eps} (matches only the empty string) 3. For all symbols a $\in \Sigma$, a is a regex with L(a) = {a} (matches only single string a) 4. Let r1, r2 be regexes. Then r1 + r2, r1r2, and r * 1 are regexes, with L(r1 + r2) = L(r1) $\cup$ L(r2), L(r1r2) = L(r1)L(r2), and L(r *1) = (L(r1))* (matches union, concatenation, and star, respectively) **For a regex to correctly represent a language L, it must match every string in L, and nothing else.**

Lecture 10 DFA（the DFA diagram, correctness of DFA, minimum number of states）Note: 每个 state 都要有 exactly |Σ| transitions，有 trapping state 等

1.DFA's Definition: D = (Q, Σ, δ, s, F)

Q: the (finite) set of states in D	Σ: the alphabet of symbols	δ:Q×Σ→Q is **the transition function**	s ∈ Q is the initial state of D	F ⊆ Q is the **set** of accepting (final) states of D

2. number of transition is |Q|*|Σ|; **each state must have exactly |Σ|** transitions leading out of it, each labelled **with a unique symbol** in Σ (only one transition)

3. Proving the correctness of a DFA with state invariant (每个 state 都有自己的 unique state invariant)

Format for proving correctness of state invariant: (structural induction)	**Format for proving minimum number of states (contradiction)**
1.Base case: Show that ε (the empty string) satisfies the state invariant of the initial state.	1.suppose for contradiction, 先 assume 我们需要比 minimal 还少一个 state
2.Induction step: For **each transition** from state q to state r on symbol a, -assume that the invariant of state q holds for some string w (I.H.) -show that the invariant of state r holds on string wa.	2.根据每个必要的 state invariant 来找找几个 strings，各 state invariant 找一个, then at least two of those strings must end at the same state (pigeonhole)
3."Postcondition": Show that the state invariant(s) of the accepting state(s) exactly describe the languages that we want the DFA to accept.	3. for **any pair（每个 pair 都要证）** of them, find some x (自己找的一个 string), that wix is NOT in L (rejected) and wjx is in L (accepted), then we have a contradiction, and what must assumed (we can find a correct DFA with 3 states) must be FALSE.
Note: state invariant 是只有这个 state 才对的东西，非常的 specific，start with 再前方的 edge, ends with incoming edge，q0 通常是 empty string	EX. {{0,1}| w has at least 3 ones}. w0 = epsilon, w1=1, w2 = 11, w3 = 111, Pair1: w0 and w1->we choose x to be 11. So w0x = 11, rejected; w1x = 111, accepted. That's what we want.
Notes on state invariants predicate p(x): **covers all cases and no overlapping** 1. All strings w reaching q must be satisfy P(w), All strings w satisfying P(w) must reach q. 2. No string should satisfy more than one invariant, 只有这个 state 才对的东西 3.Every string should satisfy one of the invariants, 非常的 specific EX. P(w): w has an even/odd number of 1's or length of string that reaches the state	Pair2: w2 and w3->No need to choose x, w2 =11 is rejected by the DFA; w3 =111 is accepted, contradiction again.

4.prove a regex is incorrect simply find a counter-example

Lecture 11 NFA (draw NFA diagram, NFA to DFA, DFA to Regex)

Note: 只要有一条 path 能通向 accepting state 就 accept, 一个 state 可以有 more than one transitions thru one symbol, 一个 state 可以没有任何 outgoing transition, 可以没有 trapping state, given a state and a symbol, it returns the set of states to which that symbol transitions, 有 ε-transition

NFA to DFA (Subset Construction-DFA 里的每个 state 都代表 a set of states in NFA)	**DFA to Regex (State Elimination)**
1.先把 NFA 的 initial state 通过 0 or more ε-transitions 可以得到的 set of states 设为 DFA 的 initial state	**1.** If the initial state q1 has incoming edges, create a new start state s and add an ε-transition to q1.
2.再从已经得到的 DFA 的 initial state 出发，每个 symbol 的 transition 都走一遍，把 set 里面的每一个 state 可以通过 symbol 得到的下一个 state 组成新的 set of state，记住**再 check 有没有通过 ε-transitions 可以得到 free state**！如果是新的 state，继续这个走每个 symbol 的 transition，一直 repeat 到没有 new states 出现 Note：如果一个 state 通过一个 symbol 去不到任何 state，那就是 empty set（也算是一个 state）	2. If there are multiple accepting states or if the final state qn has outgoing edges, create a new accepting state f and add ε-transition(s) to f from all former accepting states. (Former accepting states become non-accepting.) eliminate 一个环绕式的 state 时要确保他的 incoming 和 outgoing edge 都被 taken care 了！
	3.删掉 trapping state
	4. Eliminate state by state until only the initial and the accepting state remain.
3. Accepting states of the DFA is any states that contains an accepting state of the NFA.	(q1)—r1—(q2)—r2—(q3)变成(q1)—r1r2—(q3) (q1)—r1—(q2 自绕 r2)—r3--(q3)变成(q1)—r1(r2*)r3—(q3) 星号是 kleene star 零 or 多次重复 (q1)—r1r2 两条—(q2)变成(q1)—r1+r2—(q2) (q1)—r1—(q2)—r2—(q1)变成 q2 自绕 r1r2（eliminate 掉的 q1）

*Euclidean algorithm：GCD(a, b) = GCD(b, a % b) where b<a by precondition and a%b < b, because a%b <= b-1 by definition

	some regular expressions	**证 recurrence runtime without using repeated substitution，直接猜一个 runtime，用 induction 证. EX.**
$\sum_{k=1} k^2 = \frac{n(n+1)(2n+1)}{6}$	1. last 5th digit has to be 1 $\Rightarrow$(0+1)*1(0+1)(0+1)(0+1)(0+1)	$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + kn & if \ n > 1 \\ a & if \ n = 1 \end{cases}$ Prove by complete induction that T(n) = O(nlogn). Assume n is a power of 2.
$\sum_{k=1} k^3 = \frac{n^2(n+1)^2}{4}$	2. L = {w∈{0,1}*| w represents a binary number divisible by 2} $\Rightarrow$(0+1)*0	**We must find positive constants n0 and c such that T(n)≤cnlgn for all n ≥n0.** Choose no = 2. (一般就选让 function 满足 base case 的值，或者直接选 n=1 时，哪个好算选哪个)
$\sum_{k=1} k = \frac{n(n+1)}{2}$	3. L = {w∈{0,1}*| w starts and ends with the same symbol} $\Rightarrow$0(0+1)*0+1(0+1)*1+0+1+ε	Base case: n0 = 2, by the recursive definition, T(2) = 2a+2k. we must prove that 2a+2k ≤ cnlgn = 2c. We can do this by letting c = a + k. (as long as c ≥ k, c 不一定要完全满足这个等式, inequality 满足就行)
$\sum_{k=1} r^k = \frac{r(1-r^n)}{1-r}$	4. L = {w∈{0,1}*|w has a substring 010} $\Rightarrow$(0+1)*010(0+1)*	Inductive step: let n >2 and suppose that T(n/2) ≤ c(n/2) lg(n/2).
$\sum_{i=0}^k n^i = \frac{n^{k+1}-1}{n-1}$	**(我们只涉及得到 +，*，括号)**	T(n) = 2T(n/2) + kn ≤ 2(c(n/2) lg(n/2)) + kn = cnlg(n/2) + kn = cnlgn − cnlg2 + kn = cnlgn − cn + kn = cnlgn + (k − c)n ≤ cnlgn
$log_b(mn) = log_b(m) + log_b(n)$		Therefore, we choose c = a + k and no = 2 to complete the proof.
$log_b(^m/_n) = log_b(m) − log_b(n)$		
$log_b(m^n) = n \cdot log_b(m)$		
$a^{log_b(c)} = c^{log_b(a)}$ $log_b(I) = 0$		

Prove that if L is a regular language, then so is Rev(L).	**Iterative partial correctness**	**Karatsuba Multiplication**
Proof:根据 regular language 的 definition 来证 **Base cases:** 1. {} is a regular L. There are no strings to reverse, so rev({}) = {}, so rev({}) is a regular language. 2. By the same reason, rev({epsilon}) is a regular language. 3. {a} is a regular language, for any symbol a. But a = a^R, so rev({a}) = {a}, and so rev({a}) is regular. **Inductive case:** suppose that L and M are regular, and that rev(L) and rev(M) are regular. 1. L∪M is regular; we must prove that rev(L∪M) is regular. rev(L∪M) contains strings that are the reversal of a string in L or M. That is, Rev(L∪M) = Rev(L) ∪ Rev(M). These are regular by IH, and Rev(L) ∪Rev(M) is regular by definition. 2. LM is regular; we must prove rev(LM) is regular. rev(LM) consists of strings of the form (ab)^R, where a is in L and b is in R. Now use the identity (ab)^R = b^R a^R. That is, Rev(LM) consists of strings that are the reversal of a string from M followed by the reversal of a string in L: Rev(LM) = rev(M)rev(L); by IH, rev(L) and rev(M) are regular, and by the definition, rev(M)rev(L) is regular. 3. L* is regular; we must prove rev(L*) is regular. A string w is in rev(L*) iff w = (w1w2w3...wn)^R, where each wi is in L. But (w1w2w3...wn)^R = wn^R w_{n-1}^{R} w_{n-2}^{R} ... w3^{R}Rw2^{R}Rw1^{R}R, which is exactly those strings in rev(L*). By the IH, rev(L) is regular, so rev(L)* is regular by the recursive definition.	mystery(n): "Pre: n is an integer >= 0 Post: returns count = n // 2" count = 0 m = n while m > 1: m = m - 2 count = count + 1 return count **Prove the loop invariant m + count * 2 = n and m ≥ 0.** **Base case:** When we reach the loop, we must prove n+0*2 = n and n >= 0 which holds from the precondition. **Induction step: assume the invariant and guard are true** Assume m0+count0*2 = n and m0 >= 0 and m0 >= 2 are true, 同时满足这些条件的就是 m0 >= 2，we must prove m1+count1*2 = n and m1 >= 0 (两个部分分别证) 表示出 variable 的变化: m1 = m0-2 count1 = count0+1 First, we prove that m1 >= 0 where m1= m0-2 >= 0.m1 能进到 loop 说明至少是 2，so this is true by the loop guard. Now we prove m1+count1*2 = n, 就把 1 的 term 用之前我们得到的 m1 和 m0 的关系式代掉 m1+count1*2 =(m0-2) + (count0+1)*2 = m0 - 2 + 2count0 + 2 = m0 + count0*2 = m0 = n, by invariant **Check the post-condition: use invariant and ~loop guard** m+count*2 = n and m >= 0 and m ≤ 1 同时满足这几个式子的就是 m + count*2 = n and (m = 0 or m = 1) 有两个条件！ If m = 0, then count*2 = n (this means n is even), So, count = n//2. If m = 1, then 1+count*2 = n (this means n is odd) = n//2, as n is odd	假如 x=5678，y=1234, 把这两个数分成原本 length 的一半 n = max(len(x), len(y)) nby2 = n / 2 这个就是我们要从什么位置分 x 和 y a = x / 10^(nby2) →a=56 b = x % 10^(nby2)→b=78 c = y / 10^(nby2)→c=12 d = y % 10^(nby2)→d=34 Step1:算 ac=672 Step2:算 bd=2652 都是直接 recursion Step3: 算(a+b)(c+d)=6164 step4:算 step3-2-1=2841 Step5: prod = ac * 10^(2*nby2) + (step4*10^nby2) + bd 也就是 6720000(加 4 个 0)+2652+284000(加两个 0)=results **prove a language is not regular by proving the minimum number of states needed by the DFA is infinite.** Prove no DFA accepts L = {0^{n}1^n	n ∈ N}. We use the following infinite set of strings: 0, 00, 000, 0000, . . .; i.e. one or more 0s. We let 0j be the string of j 0s. Any DFA is required to have a finite number of states; so any DFA must place at least two of these strings into the same state. Take i < j, and suppose for contradiction that a correct DFA for L exists and puts 0i and 0j into the same state. Now, add the suffix 1i to each string, yielding 0^{i}1^i and 0^{j}1^i. The DFA puts these two strings into the same state but, as i≠ j, one of the two strings is in the language while the other is not. This is a contradiction: a state cannot be both an accepting and non-accepting state. Therefore, there is no DFA for L.