

$\neg p$	NOT	\rightarrow IF THEN	\leftrightarrow IF and only IF	\oplus XOR
V	OR	$p \rightarrow q$	$p \leftrightarrow q$	$p \oplus q$
\wedge	AND	1 1 1 1 0 0 0 1 1 0 0 1	1 1 1 1 0 0 0 1 0 0 0 1	1 1 0 1 0 1 0 1 1 0 0 0

Operator precedence: $\neg \wedge \vee \rightarrow \leftrightarrow$

Tautology always true

Consistent true for at least one scenario

Contradiction not true for any scenario
(inconsistent)

$A \equiv B$ Equivalent (has the same truth table)

De Morgan's Laws $\neg(p \wedge q) \equiv \neg p \vee \neg q$
 $\neg(p \vee q) \equiv \neg p \wedge \neg q$

$$\begin{aligned} p \rightarrow q &\equiv \neg p \vee q \\ &\equiv \neg(\neg p \wedge \neg q) \\ &\equiv \neg q \rightarrow \neg p \end{aligned}$$

\exists Some formula : for some x , the formula is true
(at least one)

"All P's are Q's" translates into $\forall x(P(x) \rightarrow Q(x))$

"No P's are Q's" translates into $\forall x(P(x) \rightarrow \neg Q(x))$

"Some P's are Q's" translates into $\exists x(P(x) \wedge Q(x))$

"Some P's are not Q's" translates into $\exists x(P(x) \wedge \neg Q(x))$

$\neg \exists x P(x) = \forall x \neg P(x)$

$\neg \forall x P(x) = \exists x \neg P(x)$

Proof techniques

↳ Direct Proof

↳ Contradiction Prove A: Assume A is false & arrive at a statement that contradicts this

↳ Contrapositive Prove $A \rightarrow B$: Prove $\neg B \rightarrow \neg A$
(if A, then B) (if not B, then not A)

Theorem 10. $n < 3^n$, for all $n \in \mathbb{N}$

Proof. Let $P(n) \equiv n < 3^n$, prove by induction that $P(n)$ is true for all n .

Basis: Prove $P(0)$ is true.

$$P(0) = 0 < 3^0 \\ = 0 < 1$$

Inductive Step: Prove $P(k) \rightarrow P(k+1)$.

Assuming $P(k) = k < 3^k$ to be true, prove that $P(k+1) = (k+1) < 3^{k+1}$ is true.

$$\begin{aligned} k < 3^k \\ k+1 < 3^k + 1 \\ k+1 < 3^k + 1 < 3^k + 3^k + 3^k \\ k+1 < 3^k + 1 < 3 \cdot 3^k \\ k+1 < 3^k + 1 < 3^{k+1} \\ k+1 < 3^{k+1} \end{aligned}$$

Conclusion: $P(k+1)$ is true, $\therefore \forall n \in \mathbb{N} n < 3^n$ is true. \square

Counting

↳ Product rule: m ways for task 1
n ways for task 2
 \vdots job can be done in $m \times n \times \dots$ ways

↳ Sum rule: job can be done in m or n or... ways \rightarrow m + n + ... ways

↳ Subtraction rule: if m and n have items in common $\rightarrow m + n - \text{common items}$
(Inclusion-Exclusion principle)

For most accounts, you need to choose a password. In this example, the password must be five to seven characters long. Each drawn from uppercase letters or digits. The password must contain at least one upper case letter.

Let's split this work by length:

Passwords	Length 5	Length 6	Length 7
All passwords (1)	36^5	36^6	36^7
No Letters (2)	10^5	10^6	10^7
Valid Passwords (1 - 2)	60 366 176	2 175 782 336	78 354 164 096
Total			80 590 312 608

How many integers less than 100 are divisible by either 2 or 3. In other words, we're talking about the cardinality of the union of the set of numbers divisible by 2 and less than 100 and the set of numbers divisible by 3 and less than 100.

However, this would be cumbersome to calculate. A simpler way is to first calculate how many numbers between 1 and 99 are divisible by 2 using $\lfloor \frac{99}{2} \rfloor = 49$. Similarly, we can calculate how many numbers between 1 and 99 are divisible by 3 using $\lfloor \frac{99}{3} \rfloor = 33$.

We must remember to decrement numbers that are divisible by both 2 and 3. Such numbers are divisible by 6, therefore $\lfloor \frac{99}{6} \rfloor = 16$.

So the answer to our original question is $49 + 33 - 16 = 66$.

Pigeonhole principle "if you have 6 pigeons and 3 holes, you'll have at least one hole with min. 2 pigeons"

A bag contains 6 blue balls, 12 red balls and 10 green balls. How many balls (minimum) must be selected to guarantee that at least 5 balls are the same colour?

$$\begin{aligned} \hookrightarrow & \# \text{boxes} = 3 \\ \hookrightarrow & \# \text{items} = 5 \\ \hookrightarrow & \text{one box with at least } 2 \text{ items} \end{aligned} \Rightarrow \boxed{\frac{13}{3}} = 5$$

comes from

$$\boxed{\frac{\# \text{items}}{\# \text{boxes}}} = \text{at least one box with at least } \times \text{ items in it}$$

Select 5 integers from the set $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$; show that at least two integers add up to 9.

$$\begin{aligned} \hookrightarrow & \# \text{boxes} = 3 \\ \hookrightarrow & \# \text{items} = 5 \\ \hookrightarrow & \text{one box with at least } 2 \text{ items} \end{aligned} \Rightarrow \boxed{\frac{5}{4}} = \textcircled{5}$$

unique
Permutation \rightarrow of n distinct items $= n!$

w/order
 \rightarrow of r elements from a set of size n : $P(n, r) = \frac{n!}{(n-r)!}$

Combination $C(n, r) = \frac{n!}{r! \times (n-r)!}$
w/o order

Alphabet $\Sigma = \{\dots\}$ set of symbols

String made from symbols
(word)

Σ^* : all possible strings incl. ϵ

$\Sigma = \{\epsilon, 0, 1\} \rightarrow \Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$

$$|\Sigma| = 2 \rightarrow |\Sigma^k| = 2^k$$

$$|\Sigma^4| = 2^4 = 16$$

Σ^+ : all non-empty strings

$\Sigma^+ = \{0, 1, 00, 01, 10, 11, \dots\}$

Σ^k : all strings of length k

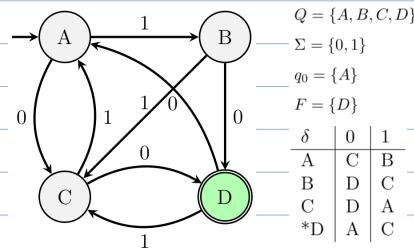
$\Sigma^2 = \{00, 01, 10, 11\}$

Finite Automaton

An automaton M is a 5-tuple² $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set called the **states**
- Σ is a finite set called the **alphabet**
- $\delta: Q \times \Sigma \rightarrow Q$ is the **transition function**
- $q_0 \in Q$ is the **start state**
- $F \subseteq Q$ is the set of **accept states**

ACCEPT if ends here
else REJECT



Language set of all strings accepted by automaton

$$L(M) = \{x \in \Sigma^* \mid M \text{ accept } x\}$$

"well behaved"

Deterministic Finite Automaton : - each state has exactly one transition per letter
(DFA)
- unique starting state

Non-deterministic Finite Automaton

if not

"too many or too few transitions"

if no transition left: REJECT!

Regular Operations

$$L_1 = \{x_1, \dots\} \quad L_2 = \{y_1, \dots\}$$

↳ Union

$$L_1 \cup L_2 = \{x_1, \dots, y_1, \dots\}$$

↳ Concatenation

$$L_1 \circ L_2 = \{x_1 y_1, x_1 y_2, \dots, x_n y_1, x_n y_2, \dots\}$$

↳ Star

$$L_1^* = \{\epsilon, x_1, \dots, x_1 x_1, x_1 x_2, \dots, x_n x_1, x_n x_2, \dots\}$$

$$A = \{\text{red, green, pink}\}$$

$$B = \{\text{apple, banana, kiwi}\}$$

$$A \cup B = \{\text{red, green, pink, apple, banana, kiwi}\}$$

$$A \circ B = \{\text{redapple, redbanana, redkiwi, greenapple, greenbanana,}\}$$

$$\text{greenkiwi, pinkapple, pinkbanana, pinkkiwi}\}$$

$$A^* = \{\epsilon, \text{red, green, pink, redred, redgreen,}\}$$

$$\text{redpink, greenred, greengreen, greenpink,}\}$$

$$\text{pinkred, pinkgreen, pinkpink, ...}\}$$

Operation	Property	Example
Union	Commutative	$A \cup B = B \cup A$
	Associative	$(A \cup B) \cup C = A \cup (B \cup C)$
	Identity	$A \cup \emptyset = A$
	Idempotence	$A \cup A = A$
	Distributive	$(A \cup B) \circ C = (A \circ C) \cup (B \circ C)$
Concatenation <small>not commutative; order is important</small>	Associative	$(A \circ B) \circ C = A \circ (B \circ C)$
	Identity	$A \circ \epsilon = A$
	Domination	$A \circ \emptyset = \emptyset$
	Distributive	$A \circ (B \cup C) = (A \circ B) \cup (A \circ C)$
Kleene Star		$\emptyset^* = \{\epsilon\}$
		$\epsilon^* = \epsilon$
		$(A^*)^* = A^*$
		$A^* A^* = A^*$
		$(A \cup B)^* = (A^* B^*)^*$

Regular Expressions

\emptyset : Empty language
any letter a : $\{a\}$ ϵ : $\{\epsilon\}$

Order of Precedence: $*, \circ, \cup$

The language of the regular expression

- What is the language of ab^* ?
— $\{a, ab, abb, abbb, \dots\}$
- What is the language of $ab^* \cup b^*$?
— $\{a, ab, abb, abbb, \dots\} \cup \{\epsilon, a, b, ab, bb, abb, bbb, \dots\} = \{\epsilon, a, b, ab, bb, abb, bbb, \dots\}$
- What is the language of $ab^+ \cup b^{+?}$?
— $\{ab, abb, abbb, \dots\} \cup \{bb, bbb, bbbb, \dots\} = ab^* \cup b^*/\{a, \epsilon, b\}$

Examples on binary alphabet $\Sigma = \{a, b\}$

- What is the language of $\Sigma^* a$?
— $\{a, aa, ba, aaa, aba, \dots\}$
— All strings ending with a
- What is the language of $\Sigma^* a \Sigma^*$?
— $\{a, aa, ab, ba, aaa, aab, aba, abb, baa, bab, bba, \dots\}$
— All strings containing at least one a .

All binary words containing bb $\rightarrow \Sigma^* bb \Sigma^*$
 $\{bb, abb, bba, bbb, aabb, abba, abbb, \dots\}$

All binary words ending with ab or ba $\rightarrow (\Sigma^* ab) \cup (\Sigma^* ba)$
 $\{ab, ba, aab, aba, bab, bba, aaab, aaba, \dots\}$

All binary words with at most one a $\rightarrow (b^* ab^*) \cup b^*$
 $\{\epsilon, a, b, ab, ba, bb, abb, bba, bab, bba, \dots\}$

All binary strings of length 3 $\rightarrow \Sigma \Sigma \Sigma$
 $\{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

All binary strings of length at least 3 $\rightarrow \Sigma \Sigma \Sigma^+$

All binary strings of length at most 3 $\rightarrow \epsilon \cup \Sigma \cup \Sigma \Sigma \cup \Sigma \Sigma \Sigma$

Kleene's Theorem Language: L

is regular \leftrightarrow IFF can be expressed as finite automaton

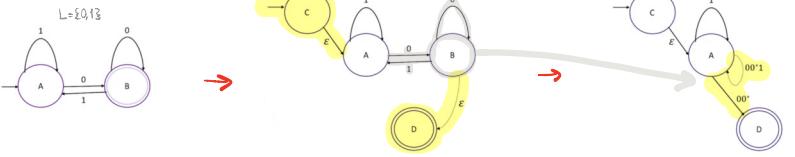
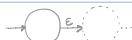
There is a regen: R
 $L(R) = L$

There is a finite automaton: A
 $L = L(A)$

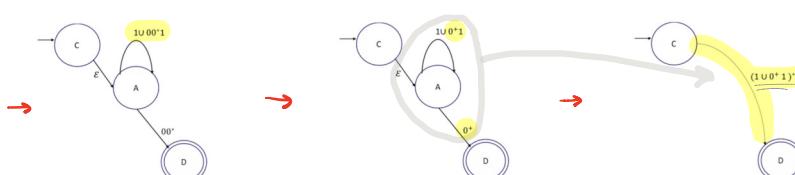
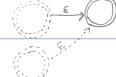
Convert: Finite Automaton \rightarrow Regular Expression

0- transition labels \rightarrow language

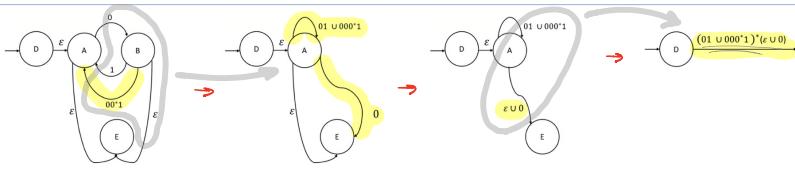
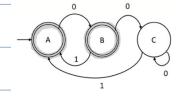
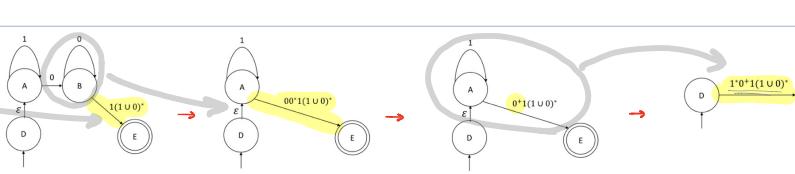
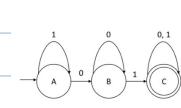
1- Create initial state to first state with ϵ -transition



2- Create final state from existing final state(s) with ϵ -transition



3- Remove all states between initial and final state
one-by-one from last to first



Closure properties

Theorem 15. If L_1 and L_2 are regular languages on alphabet Σ , then the following languages are also regular:

- $U - L_1$: This means $\Sigma^* - L_1$ or the complement of L_1 : $\neg L_1$
- $L_1 \cup L_2$: The union of L_1 and L_2
- $L_1 \cap L_2$: The intersection of L_1 and L_2
- $L_1 L_2$: The product of L_1 and L_2
- L_1^* : The Kleene star of L_1

Prove: L is non-regular:

1- Assume L is regular

2- Find contradiction w/ Closure properties

only regular languages can be represented by Finite Automata
irregular languages need context-free Grammar

Context-free Grammar

A context-free grammar G is a 4-tuple (V, Σ, R, S) where:

Variables (V)

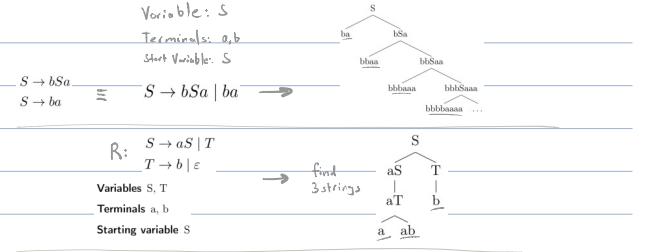
Terminals (Σ)

Rules (R)

Start Variable (S)

Generating strings

- Starting from start symbol, read its rule
- Find a variable in the rule of the start symbol and replace it with the rule
- Repeat step 2 until there are no variables left



Regular Expr. to CFG

Example 1

ab^* to CFG

CFG:

$$\begin{aligned} ab^* &\rightarrow b^*: U \rightarrow bU | \epsilon \\ ab^*: S \rightarrow aU & \Rightarrow S \rightarrow aU \\ ab^*: S \rightarrow aU & \Rightarrow U \rightarrow bU | \epsilon \end{aligned}$$

Example 2

$ab^* \cup b^*$

$$\begin{aligned} b^*: U \rightarrow bU | \epsilon & \Rightarrow S \rightarrow aU \\ b^*: S \rightarrow aU & \Rightarrow U \rightarrow bU | \epsilon \end{aligned}$$

Union
 $aUb \rightarrow aUb$

$$\begin{aligned} S \rightarrow aU & \mid U \\ U \rightarrow bU & \mid \epsilon \end{aligned}$$

$S \rightarrow aTS | aS | \epsilon$
 $T \rightarrow SbT | ba$

- Show a derivation of string $aabb$
 $S \rightarrow aTS \Rightarrow aSbTS \Rightarrow aaSbTS \Rightarrow aabTS \Rightarrow aabbS \Rightarrow aabb$
- Show a derivation of string $abaabaaba$
 $S \rightarrow aTS \Rightarrow abaS \Rightarrow abaaTS \Rightarrow abaaaS \Rightarrow abaaaTS \Rightarrow abaabaabaS \Rightarrow abaabaaba$
- Show a derivation of string $abaabbbba$
 $S \rightarrow aTS \Rightarrow abaS \Rightarrow abaaTS \Rightarrow abaabTS \Rightarrow abaabSbTS \Rightarrow abaabbTS \Rightarrow abaabbbaS \Rightarrow abaabbbba$

Example 3

$ab^+ \cup b^+b$

CFG:

$$\begin{aligned} b^+: U \rightarrow bU | b & \Rightarrow S \rightarrow aU | T \\ b^+b: T \rightarrow Ub & \Rightarrow T \rightarrow Ub \\ ab^+: S \rightarrow aU & \Rightarrow U \rightarrow bU | b \end{aligned}$$

Simplified

$$\begin{aligned} S \rightarrow aU & \mid bU \\ U \rightarrow bU & \mid b \end{aligned}$$

Example 4

$\Sigma^* a \Sigma^*, \Sigma = \{a, b\}$

$$\begin{aligned} \sum^*: U \rightarrow aU | bU | \epsilon & \Rightarrow S \rightarrow UaU \\ \sum^* \sum^*: S \rightarrow aUaU & \Rightarrow U \rightarrow aU | bU | \epsilon \end{aligned}$$

CFG:

$$\begin{aligned} S \rightarrow UaU & \Rightarrow U \rightarrow aU | bU \\ U \rightarrow aU | bU & \mid \epsilon \end{aligned}$$

Example 5

$\Sigma\Sigma\Sigma^+$
binary strings of length at least 3

$$\begin{aligned} \Sigma^+ = (aUb)^+: U \rightarrow aU | bU | a | b & \Rightarrow S \rightarrow aV | bV \\ \Sigma\Sigma^+: V \rightarrow aU | bU & \Rightarrow V \rightarrow aU | bU \\ \Sigma\Sigma\Sigma^+: S \rightarrow aV | bV & \Rightarrow U \rightarrow aU | bU | a | b \end{aligned}$$

Simplified

$$S \rightarrow aaU | abU | buU | bbU$$

Turing Machines

A Turing Machine consists of $(Q, \Sigma, \Gamma, \delta, q_1, q_{Acc}, q_{Rej})$, where:

TH A Tape Head

Q A finite set of states

Σ The input alphabet where $\Sigma \subseteq \Gamma$

Γ The tape alphabet, including the blank symbol ■

δ A transition function $\delta: Q \times \Gamma \rightarrow (Q \times \Gamma \times \{L, R\})$

q_1 The start state, where $q_1 \in Q$

q_{Acc} Accept state } not essential for TMs

q_{Rej} Reject state }

Transition Function

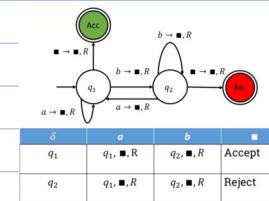
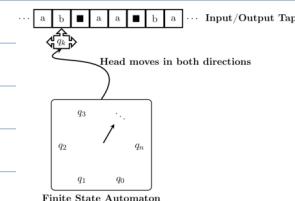
Input: - current state
- read letter

Output: - next state
- letter to write
- direction to move (L or R)

TM's:

- terminates when reaching Accept/Reject (DFA's may pass through) \rightarrow accepts/rejects original string on tape (not final changed string)
- may enter infinite loop

Finite State Automaton



The Language of TMs

The language of a Turing Machine, M , is composed of all the strings accepted by M . Formally, $\mathcal{L}(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

- If $w \in M$, M reaches accept state
- If $w \notin M$, M does not reach accept state
 - either reaches reject state
 - or enters infinite loop
- A language is recognizable if it is accepted by a Turing Machine
- A Turing Machine that recognizes $\mathcal{L}(M)$ is called a Recognizer
- Recursively Enumerable (RE) is the class of all recognizable languages
- A Turing Machine that never enters an infinite loop is called a Decider
- A language accepted by a decider is Decidable
- R is the class of all decidable languages

Halting problem

- Every decider is a Turing Machine
- $R \subseteq RE$
- Halting Problem: Given a Turing Machine, does it halt?

Church-Turing Thesis states that the Halting Problem is an undecidable problem.

What we know

- Every regular language is context-free (FSA and RE)
- Every context-free language is Turing-decidable (CFGs)
- Every decidable language is recognisable (Decide Turing Machine)

Chomsky Hierarchy

Grammar	Languages	Automaton	Example
Type-0	Recursively enumerable	Turing machine	
Type-1	Context-sensitive	Turing machines with bounded tape	$a^n b^n c^n$
Type-2	Context-free	Push-down	$a^n b^n$
Type-3	Regular	Finite state	$a^* b^*$

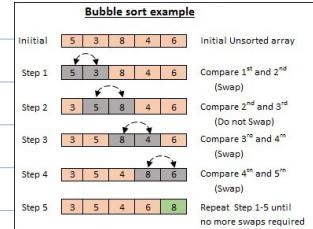
Insertion Sort

- Pick the first item in the unordered part of the list
- Compare with the previous items (ordered)
- Move the item if necessary.

First Pass	$[23 \ 1 \ 10 \ 5 \ 2]$	$[23 \ 1 \ 10 \ 5 \ 2]$
Second Pass	$[23 \ 1 \ 10 \ 5 \ 2]$	$[1 \ 23 \ 10 \ 5 \ 2]$
Third Pass	$[1 \ 23 \ 10 \ 5 \ 2]$	$[1 \ 10 \ 23 \ 5 \ 2]$
Fourth Pass	$[1 \ 10 \ 23 \ 5 \ 2]$	$[1 \ 5 \ 10 \ 23 \ 2]$
Fifth Pass	$[1 \ 5 \ 10 \ 23 \ 2]$	$[1 \ 2 \ 5 \ 10 \ 23]$

- $i = 2$
- While i does not exceed the length of the list:
 - Select the i^{th} entry of the list as a pivot entry
 - Move the pivot entry to a temporary position; leave a hole in the list
 - While there is an item above the hole and item $>$ pivot entry:
 - Move the item to the hole, leaving a hole above the item
 - Move the pivot entry into the hole.
- $i = i+1$

Bubble Sort



End = length of the List -1

```

While End>1:
  i = 1
  While i does not exceed End:
    If List[i]>List[i+1]:
      Swap List[i] and List[i+1]
    i = i+1
  End = End-1
Return List
  
```


Stable Matching "Algorithm of Happiness"

Problem: n hospitals and n medical students to be matched
each w/ preferences

Matching M: (h,s) each h and s appears at most one pair M

Perfect Match each h and s appears at exactly one pair M

a pair is **unstable** when $\begin{cases} \text{student } s \text{ prefers hospital } h \text{ over his assigned hospital} \\ \text{AND} \\ \text{hospital } h \text{ prefers student } s \text{ over its assigned student} \end{cases}$

to find **stable** matches:

Gale-Shapley Algorithm, 1962

1. Each unmatched hospital offers a place to a student on the top of its list
2. Students
 - With one offer** accept the offer
 - With more than one offer** accept top hospital that made them an offer
3. Repeat until all hospitals are matched

Time Complexity

	Sequential Search	Binary Search	Insertion Sort	Bubble Sort	Quick Sort	Merge Sort
Worst case	n	$\log n$	$\frac{n \cdot (n-1)}{2}$	$\frac{n \cdot (n-1)}{2}$	n^2	$n \cdot \log n$
Average case	$\frac{n}{2}$	$\log n$	$\frac{n \cdot (n-1)}{4}$	n^2	$n \cdot \log n$	$n \cdot \log n$
Best case	1	1	$n-1$	$n-1$	$n \cdot \log n$	$n \cdot \log n$

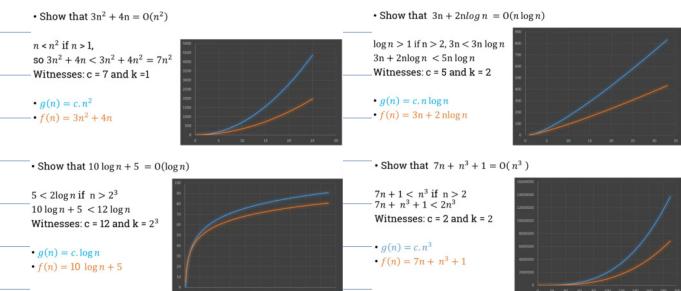
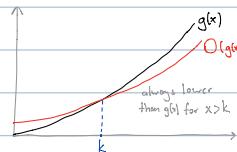
Asymptotic Behavior growth of the (time) function when input size grows rapidly

To determine: Find the fastest growing term in function & ignore smaller terms!

$$\begin{aligned} \text{Ex: } & 5n^3 + 7n^2 - 8n & 5n + n\log n \\ & n^3 + 7n^2 \log n & n + 7\log n \\ & 10n^2 + n^2 \log n + 2n \end{aligned}$$

Asympt. Behavior	Graph
Constant	horizontal line
Logarithmic	log. curve
Linear	straight line
Quadratic	parabolic
Higher order polynomial	
Exponential	

Big O notation $O(g(x)) \leq c \cdot g(x)$ where $\begin{cases} c \text{ and } k \text{ are constant (called witnesses)} \\ \text{AND} \\ x > k \end{cases}$



Recursion Complexity - Guess & prove by induction

- Tree method

- Master Theorem

The recurrence $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$, $a \geq 1, b > 1, d \geq 0$

$$d < \log_b^a \rightarrow T(n) = O(n^{\log_b^a})$$

$$d = \log_b^a \rightarrow T(n) = O(n^d \log n)$$

$$d > \log_b^a \rightarrow T(n) = O(n^d)$$

Master Theorem - example

$$\begin{aligned} & T(n) = T\left(\frac{n}{2}\right) + O(1), a = 1, b = 2, d = 0 \\ & 0 = \log_2^1 \\ & T(n) = O(n^0 \log n) \text{ so } T(n) = O(\log n) \end{aligned}$$

$$\begin{aligned} & T(n) = 3T\left(\frac{n}{4}\right) + O(1), a = 3, b = 4, d = 0 \\ & 0 < \log_4^3 \\ & T(n) = O(n^{\log_4^3}) \text{ so } T(n) = O(n^{\log_4^3}) \end{aligned}$$