(biconditional) (exclusive or)

$\neg p$ NOT → IF THEN ↔ IF and only IF ⊕ XOR

$\vee$ OR

$\wedge$ AND

p	q	$p \to q$
1	1	1
1	0	0
0	1	1
0	0	1

p	q	$p \leftrightarrow q$
1	1	1
1	0	0
0	1	0
0	0	1

p	q	$p \oplus q$
1	1	0
1	0	1
0	1	1
0	0	0

operator precedence: $\neg$ $\wedge$ $\vee$ $\to$ $\leftrightarrow$ (First → Last)

Tautology always true

Consistent true for at least one scenario

Contradiction (inconsistent) not true for any scenario

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

De Morgan's Laws

$\neg(p \wedge q) \equiv \neg p \vee \neg q$

$\neg(p \vee q) \equiv \neg p \wedge \neg q$

$p \to q \equiv \neg p \vee q$

$\equiv \neg(p \wedge \neg q)$

$\equiv \neg q \to \neg p$

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

$\forall x$ some formula : for all x, the formula is true

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

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

"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 $A \to B$ (if A, then B): Prove $\neg B \to \neg A$ (if not B, then not A)

↳ **Induction**

1) Base case: Prove that P(1) is true

2) Inductive step: > Assume that P(k) is true

> Prove that P(k+1) is true

3) Conclusion ∵ $\forall n \in \mathbb{N}\, P(n)$ is true □

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

Proof. Let $P(n) = n < 3^n$, 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 $P(k) \to P(k+1)$.

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

$$k < 3^k$$
$$k+1 < 3^k + 1$$
$$k+1 < 3^k + 1 < 3^k + 3^k + 3^k$$
$$k+1 < 3^k + 1 < 3 \cdot 3^k$$
$$k+1 < 3^k + 1 < 3^{k+1}$$
$$k+1 < 3^{k+1}$$

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

Counting

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

↳ **Sum rule:** job can be done in m or n or ... ways → it can be done in $m+n+\dots$ ways

↳ **Subtraction rule:** if m and n have items in common → $m+n-$ 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 priciple

"if you have 4 pigeons and 3 holes, you'll have at least one hole with min. 2 pigeons"

$$\left\lceil \frac{\#\text{items}}{\#\text{boxes}} \right\rceil = \text{at least one box with at least } X \text{ items in it}$$

(ceiling: round up)

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?

↳ #boxes = 3 (r, g, b)
↳ one box with at least = 5 $\Rightarrow \left\lceil \frac{13}{3} \right\rceil = 5$

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.

1+8 = 9, 2+7 = 9, ... (a, b, c, d, d, c, b, a)

↳ #boxes = 4 (a, b, c, d)
↳ #items = 5 $\Rightarrow \left\lceil \frac{5}{4} \right\rceil = 2$

Permutation

unique w/order

↳ of n distinct items $= n!$

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

Combination

w/o order

$$C(n,r) = \frac{n!}{r! \times (n-r)!}$$

Alphabet

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

String (word) made from symbols

Σ^* : all possible strings incl. ε — $\Sigma = \{0,1\} \rightarrow \Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \dots\}$ — $|\Sigma| = 2 \rightarrow |\Sigma^k| = 2^k$, $|\Sigma^6| = 2^6 = 64$

Σ^+ : all non-empty strings — $\Sigma^+ = \{0, 1, 00, 01, 10, 11, \dots\}$

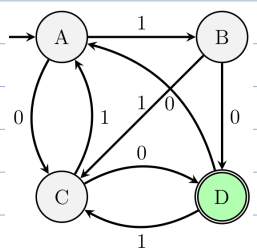
Σ^k : all strings of length k — $\Sigma^2 = \{00, 01, 10, 11\}$

Finite Automaton

An automaton M is a 5-tuple[2] $(Q, \Sigma, \delta, q_0, F)$ 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



$Q = \{A, B, C, D\}$
$\Sigma = \{0, 1\}$
$q_0 = \{A\}$
$F = \{D\}$

δ	0	1
A	C	B
B	D	C
C	D	A
*D	A	C

Language

set of all strings accepted by automaton

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

"well behaved"

Deterministic Finite Automaton (DFA)

- each state has exactly one transition per letter
- unique starting state

↳ if not

Non-deterministic Finite Automaton (NFA)

"too many or too few transitions"

↳ if no transition left: REJECT!

Regular Operations

$L_1 = \{x, \dots\}$ $L_2 = \{y, \dots\}$

↳ Union $L_1 \cup L_2 = \{x, \dots, y, \dots\}$

↳ Concatenation $L_1 \circ L_2 = \{x_1y_1, x_1y_2, x_2y_1, \dots\}$

↳ Star $L_1^* = \{\varepsilon, x_1, \dots, x_1x_1, x_1x_2, \dots, x_1x_1x_1, x_1x_2x_2, \dots\}$

$A = \{red, green, pink\}$
$B = \{apple, banana, kiwi\}$
$A \cup B = \{red, green, pink, apple, banana, kiwi\}$
$A \circ B = \{redapple, redbanana, redkiwi, greenapple, greenbanana, greenkiwi, pinkapple, pinkbanana, pinkkiwi\}$
$A^* = \{\varepsilon, red, green, pink, redred, redgreen, redpink, greenred, greengreen, greenpink, pinkred, pinkgreen, pinkpink, \dots\}$

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 (not commutative! header is important)	Associative	$(A \circ B) \circ C = A \circ (B \circ C)$
	Identity	$A \circ \varepsilon = A$
	Domination	$A \circ \emptyset = \emptyset$
	Distributive	$A \circ (B \cup C) = (A \circ B) \cup (A \circ C)$
Kleene Star		$\emptyset^* = \{\varepsilon\}$
		$\varepsilon^* = \varepsilon$
		$(A^*)^* = A^*$
		$A^* \circ A^* = A^*$
		$(A \cup B)^* = (A^* B^*)^*$

Regular Expressions

$\emptyset$: Empty language

any letter a: $\{a\}$ ε: $\{\varepsilon\}$

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

The language of the regular expression

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

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

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

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

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

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

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

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

All binary strings of length at most 3 $\longrightarrow \varepsilon \cup \Sigma \cup \Sigma\Sigma \cup \Sigma\Sigma\Sigma$

Kleene's Theorem

Language: L

is regular ⇄ IFF ⇄ can be expressed as finite automaton

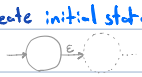
There is a regex: R, L(R) = L

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

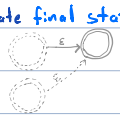
Convert: Finite Automaton → Regular Expression

0- transition labels → language

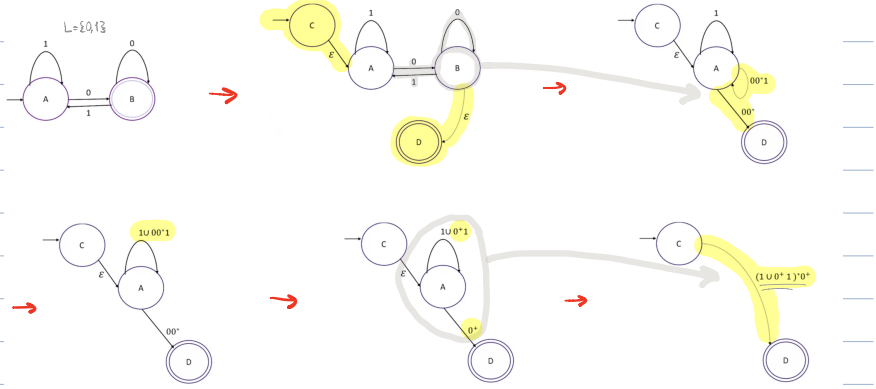
1- Create initial state to first state with ε-transition

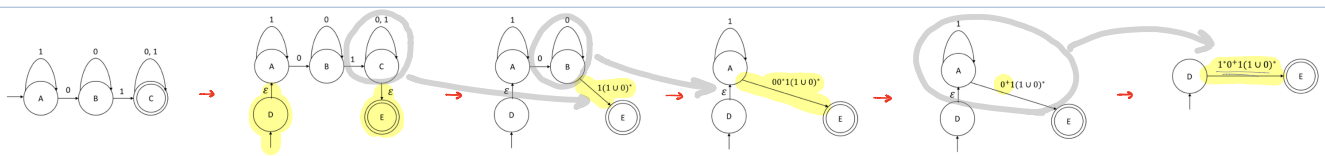


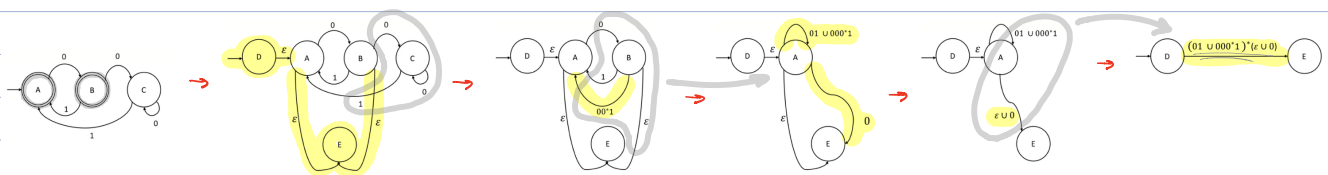
2- Create final state from existing final state(s) with ε-transition



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 Langs can be represented by Finite Automata
↳ irregular Langs need context-free Grammar

Context-free Grammar

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

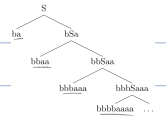
- **Variables (V)**
- **Terminals (Σ)**
- **Rules (R)**
- **Start Variable (S)**

Generating strings

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

$S \to bSa$
$S \to ba$
$\equiv$ $S \to bSa \mid ba$

Variable: S
Terminals: a, b
Start Variable: S



R: $S \to aS \mid T$
$T \to b \mid \varepsilon$

Variables S, T
Terminals a, b
Starting variable S

→ find 3 strings

Regular Expr. to CFG

Union
$a \cup b \to a \mid b$

Example 1

ab^* to CFG

b^*: $U \to bU \mid \varepsilon$
ab^*: $S \to aU$

⇒ CFG:

$S \to aU$
$U \to bU \mid \varepsilon$

Example 2

$ab^* \cup b^*$

b^*: $U \to bU \mid \varepsilon$
ab^*: $S \to aU$

⇒ CFG:

$S \to aU \mid U$
$U \to bU \mid \varepsilon$

$S \to aTS \mid aS \mid \epsilon$
$T \to SbT \mid ba$

- Show a derivation of string *aabba*
  $S \Rightarrow aTS \Rightarrow aSbTS \Rightarrow aaSbTS \Rightarrow aabTS \Rightarrow aabbaS \Rightarrow aabba$
- Show a derivation of string *abaabaaba*
  $S \Rightarrow aTS \Rightarrow abaS \Rightarrow abaaTS \Rightarrow abaabaS \Rightarrow abaabaaTS \Rightarrow abaabaabaS \Rightarrow abaabaaba$
- Show a derivation of string *abaabbbba*
  $S \Rightarrow aTS \Rightarrow abaS \Rightarrow abaaTS \Rightarrow abaaSbTS \Rightarrow abaabTS \Rightarrow abaabSbTS \Rightarrow abaabbTS \Rightarrow abaabbSbTS \Rightarrow abaabbbTS \Rightarrow abaabbbbaS \Rightarrow abaabbbba$

Example 3

$ab^+ \cup b^+b$

b^+: $U \to bU \mid b$
b^+b: $T \to Ub$
ab^+: $S \to aU$

⇒ CFG:

$S \to aU \mid T$
$T \to Ub$
$U \to bU \mid b$

simplified ⇓

$S \to aU \mid uB$
$U \to bU \mid b$

Example 4

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

Σ^*: $U \to aU \mid bU \mid \varepsilon$
$\Sigma^* a \Sigma^*$: $S \to UaU$

⇒ CFG:

$S \to UaU$
$U \to aU \mid bU \mid \varepsilon$

Example 5

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

$\Sigma^+ = (a \cup b)^+$: $U \to aU \mid bU \mid a \mid b$
$\Sigma\Sigma^+$: $V \to aU \mid bU$
$\Sigma\Sigma\Sigma^+$: $S \to aV \mid bV$

⇒ CFG:

$S \to aV \mid bV$
$V \to aU \mid bU$
$U \to aU \mid bU \mid a \mid b$

simplified ⇓

$S \to aaU \mid abU \mid baU \mid bbU$

Turing Machines

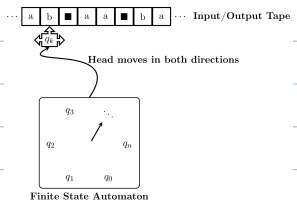
a Turing Machine consists of $(Q, \Sigma, \Gamma, \delta, q_1, q_{Acc}, q_{Rej})$, where:

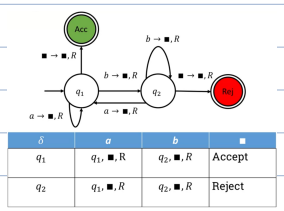
- **TH** A Tape Head
- **Q** A finite set of states
- **Σ** The input alphabet where $\Sigma \subseteq \Gamma$
- **Γ** The tape alphabet, including the blank symbol ■
- **δ** A transition function $\delta : Q \times \Gamma \to (Q \times \Gamma \times \{L, R\})$
- q_1 The start state, where $q_1 \in Q$
- q_{Acc} Accept state
- q_{Rej} Reject state

} not essential for TMs

Transition Function
Input: - current state
- read letter

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





δ	a	b	■
q_1	$q_1, ■, R$	$q_2, ■, R$	Accept
q_2	$q_1, ■, R$	$q_2, ■, R$	Reject

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

The Language of TMs

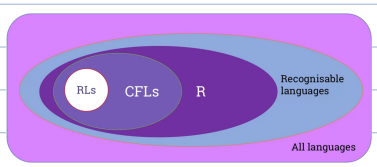
The language of a Turing Machine, M, is composed of all the strings accepted by M. Formally, $\mathcal{L}(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

- 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
- $R \subset RE$
- Halting Problem: Given a Turing Machine, does is halt?

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



What we know

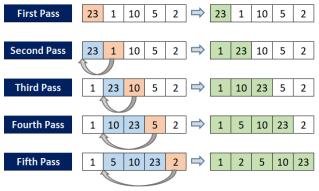
- Every regular language is context-free (FSA and RE)
- Every context-free languages is Turing-decidable (CFGs)
- Every decidable language is recognisable (Decide Turing Machine)

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	$a^* b^*$

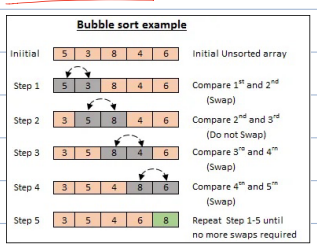
Insertion Sort

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



- 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

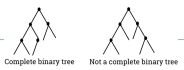


End = length of the List -1
While End>1:
  i = 1
  While i does not exceed End:
    If List[i] > List[i+1]:
      Swap List[i] and List[i+1]
    i = i+1
  End = End-1
Return List

Heap Sort

Binary tree: rooted tree where every vertex has at most 2 children

Complete Binary tree: – every node has exactly 2 children except the lowest level
– all leaves are as far to the left as possible



Max Heap: complete binary tree where each internal node's value is ≥ its children

Min Heap: " " " " " " " " " " ≤ " "

(Min) Heap Sort

1- Turn list into complete binary tree

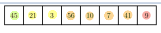


Figure 30: List to be represented as Binary Tree

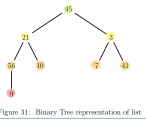


Figure 31: Binary Tree representation of list

2- Heapify (transform into heap tree)

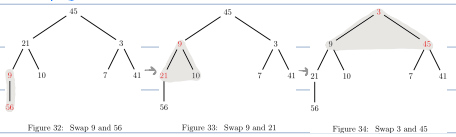


Figure 32: Swap 9 and 56
Figure 33: Swap 9 and 21
Figure 34: Swap 3 and 45

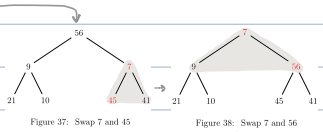


Figure 37: Swap 7 and 45
Figure 38: Swap 7 and 56

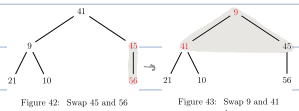


Figure 42: Swap 45 and 56
Figure 43: Swap 9 and 41

3- Remove the top node from tree into sorted list
and move bottom right leaf into root

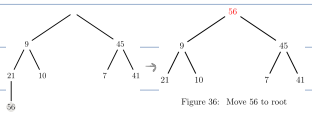


Figure 35: Remove 3
Figure 36: Move 56 to root

& Repeat 2

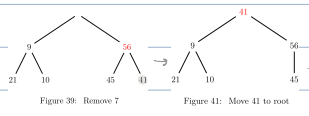


Figure 39: Remove 7
Figure 41: Move 41 to root

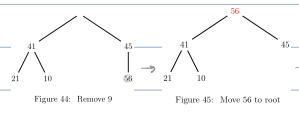
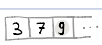


Figure 44: Remove 9
Figure 45: Move 56 to root


Figure 57: Remove 45
Figure 58: Move 56 to root



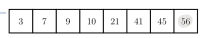


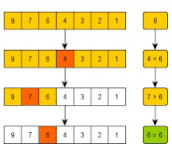
Figure 59: Sorted list after Heap Sort

Finding element in a sorted list:

Sequential Search

Binary Search

– Start in the middle and compare
– Halve the list
– Proceed to the promising half
– Repeat



```
def search(List, item):
  If List is Empty:
    Report search has failed
  Else:
    Pivot = middle entry of the list
    If Pivot == Item:
      Report search has succeeded
    Else If Pivot > Item:
      List = Items preceding Pivot
      search(List, item)
    Else:
      List = Items following pivot
      search(List, item)
```

recursive sorts

Quicksort

1- Select a pivot element (middle element in our example)

2- Group all other elements into 2 groups: lower values & higher values

3- Run Quicksort on the groups & return Quicksort(LowerGroup) + Pivot + Quicksort(HigherGroup) as sorted list

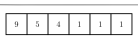


Figure 60: Unsorted array


Figure 61: Pivot


Figure 62: Sub-vectors


Figure 63: New pivots


Figure 64: Split into smaller vectors

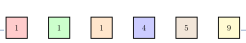
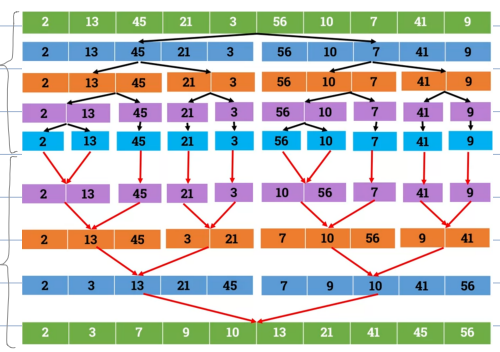


Figure 65: More pivots

```
def QuickSort(List):
  If List has one item:
    Return List
  Else:
    Pivot = the middle entry of the List
    Delete Pivot from the List
    For item in List:
      If Pivot > Item:
        ListLeft.append(item)
      Else:
        ListRight.append(item)
    C = QuickSort(ListLeft) + Pivot + QuickSort(ListRight)
    Return C
```

Mergesort

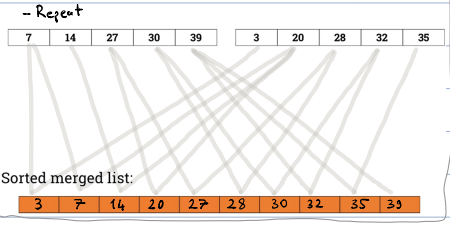
Split list into groups until broken into 1-entry lists
↓
each sorted by default!



Merge pairs of sorted lists until reaching one sorted list

Merge two sorted lists into one sorted list (iterative)

– Compare first entries of each list
– Move the smaller entry into the sorted merged list
– Repeat




```
def MergeSort(List):
  N = size of the List
  if N = 1:
    return List
  ListLeft = List[1..ceiling(N/2)]
  ListRight = List[ceiling(N/2)+1..N]
  ListLeft = MergeSort(ListLeft)
  ListRight = MergeSort(ListRight)
  return Merge(ListLeft, ListRight)
```

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 { student s prefers hospital h over his assigned hospital **AND** hospital h prefers student s over its assigned student

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	$\log_2 n$	$\frac{n\cdot(n-1)}{2}$	$\frac{n\cdot(n-1)}{2}$	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!

Ex: $5n^3+7n^2-8n$ $\quad 5n+n\log n$

$n^3+7n^2\log n$ $\quad n+7\log n$

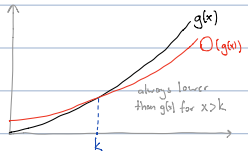
$10n^2+n^2\log n+2n$

Asymp. Behavior	Graph
Constant	horizontal line
Logarithmic	log. curve
Linear	straight line w/slope
Quadratic	parabolic
Higher order polynomial	
Exponential	

complexity

Big O notation

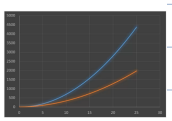
$O(g(x)) \le c\cdot g(x)$ where { c and k are constant (called witnesses) **AND** $x > k$



- Show that $3n^2+4n=O(n^2)$

$n<n^2$ if $n>1$,
so $3n^2+4n<3n^2+4n^2=7n^2$
Witnesses: c = 7 and k = 1

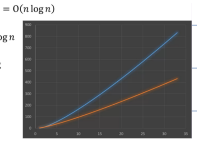
- $g(n)=c.n^2$
- $f(n)=3n^2+4n$



- Show that $3n+2n\log n=O(n\log n)$

$\log n>1$ if $n>2$, $3n<3n\log n$
$3n+2n\log n<5n\log n$
Witnesses: c = 5 and k = 2

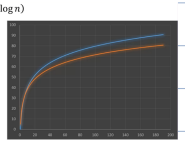
- $g(n)=c.n\log n$
- $f(n)=3n+2n\log n$



- Show that $10\log n+5=O(\log n)$

$5<2\log n$ if $n>2^3$
$10\log n+5<12\log n$
Witnesses: c = 12 and k = 2^3

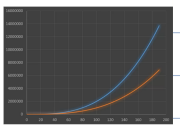
- $g(n)=c.\log n$
- $f(n)=10\log n+5$



- Show that $7n+n^3+1=O(n^3)$

$7n+1<n^3$ if $n>2$
$7n+n^3+1<2n^3$
Witnesses: c = 2 and k = 2

- $g(n)=c.n^3$
- $f(n)=7n+n^3+1$



Recursion Complexity

- Guess & prove by **induction**
- **Tree** method
- **Master Theorem**

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

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

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

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

Master Theorem – example

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

- $T(n)=3T\left(\frac{n}{4}\right)+O(1)$, a = 3, b = 4, d = 0
- $0<\log_4^3$
- $T(n)=O(n^{\log_b^a})$ so $T(n)=O(n^{\log_4^3})$