Correctness by Design

Formal Languages

FSAs

Notes
Merge Sort

Previously...

MergeSort(A, b, e):
    if b == e:
        return
    m = (b + e) // 2
    MergeSort(A, b, m)
    MergeSort(A, m+1, e)
    for i = b, ..., e:
        B[i] = A[i]
    c = b
    d = m + 1
<snip>
Correctness by Design

Merge loop
Correctness by Design

Merge loop
Correctness by Design

Merge loop
# LI: $A[b..i-1]$ contains $B[b..c-1]$ and $B[m+1..d-1]$, sorted, and $A[b..i-1] \leq B[c..m]$, $B[d..e]$

for $i = b, \ldots, e$:
    if $d > e$ or ($c \leq m$ and $B[c] < B[d]$):
        $A[i] = B[c]$
        $c = c + 1$
    else:  # $d \leq e$ and ($c > m$ or $B[c] \geq B[d]$)
        $A[i] = B[d]$
        $d = d + 1$
Correct?
Merge Sort

# Pre: 0 <= b <= e < len(A), A[b..e] comparable
MergeSort(A, b, e):
    if b == e: return
    m = (b + e) // 2
    MergeSort(A, b, m)
    MergeSort(A, m+1, e)
    # LI: left as an exercise!
    for i = b, ..., e: B[i] = A[i]
    # loop post: B[b..e] = A[b..e]
    c = b
d = m + 1
    # LI: A[b..i-1] contains B[b..c-1] and B[m+1..d-1],
    # sorted, and A[b..i-1] <= B[c..m], B[d..e]
    for i = b, ..., e:
        if d > e or (c <= m and B[c] < B[d]):
            A[i] = B[c]
            c = c + 1
        else: # d <=e and (c > m or B[c] >= B[d])
            A[i] = B[d]
            d = d + 1
    # loop post: A[b..e] contains B[b..m] and B[m+1..e], sorted
# Post: A[b..e] sorted (ie. contains same elements as before,
#       reordered so that A[b] <= ... <= A[e])
Many problems can be reduced to languages: logical formulas, identifiers for compilation, natural language processing. Key question is recognition:

Given language $L$ and string $s$, is $s \in L$?
Ways to Express Languages

- Descriptive (regular expressions, context-free grammars)

- Procedural (finite state automata, pushdown automata)
Definitions and Notation

Some definitions:

**alphabet:** finite, non-empty set of symbols. Conventionally denoted $\Sigma$.

**string:** finite (including empty) sequence of symbols over an alphabet. Convention: $\varepsilon$ is the empty string, never an allowed symbol,

**language:** any set of strings over some alphabet $\Sigma$. Possibly empty, possibly infinite subset.

**Note:** $\emptyset \neq \{\varepsilon\}$. 
Definitions and Notation

Operations on strings

\[ |s|: \text{ length of } s, \, \text{the number of symbols in } s \]

\[ s = t: \text{ if and only if } |s| = |t|, \text{ and } s_i = t_i \text{ for } 1 \leq i \leq |s|. \]

\[ s^R: \text{ reversal of } s \text{ is obtained by reversing symbols of } s \]
Definitions and Notation

Operations on strings

$st$ or $s \circ t$: concatenation of $s$ and $t$ — all characters of $s$ followed by all those of $t$

$s^k$: denotes $s$ concatenated with itself $k$ times

$\Sigma^n$: all strings of length $n$ over $\Sigma$, $\Sigma^*$ denotes all strings over $\Sigma$. 
Operations on Languages

For all languages, \( L, L' \) over alphabet \( \Sigma \):

\[ \overline{L} : \text{Complement of } L, \text{i.e. } \Sigma^* - L. \]

\[ L \cup L' : \text{union} \]

\[ L \cap L' : \text{intersection} \]

\[ L - L' : \text{difference} \]
Finite State Automata (FSAs)
aka Finite State Machines (FSMs)

Simple models of computing devices to analyze strings.

Consist of states and transitions between states

Fixed, finite number of states - includes an “initial” state, and some “accepting” or “final” states

“Accept” or “reject” a string, depending on the last state
Example: Subway Turnstile Finite State Machine

What are the rules for turnstiles?
Example: Float Machine

What strings are floats in Python?
Example: States needed to classify a string

What state is a stingy vending machine in, based on coins?
Accepts only nickles, dimes, and quarters,
No change given, everything costs 30 cents.

Transition table:
Example: Integer multiples of 3