CSC236 Notes

Induction

Predicate: A logical statement $P\colon \mathbb{N} \to \{\text{True}, \text{False}\}$ based on the value of a variable, usually n.

	Simple Induction	**Strong/Complete Induction**
Base Case	$P(\alpha)$	$P(\alpha_1), \dots P(\alpha_n)$
Inductive Step	$\forall k \geq \alpha, P(k) \Rightarrow P(k+1)$	$\forall k > \alpha_n, \big(\forall k' < k, P(k')\big) \Rightarrow P(k)$

Well-Ordering Principle (WOP/PWO): Every non-empty subset of $\mathbb{N}$ contains a minimum.

$$\forall S \subseteq \mathbb{N}, S \neq \emptyset \Rightarrow \exists m \in S, \forall s \in S, m \leq s$$

- Simple induction, Strong induction, and WOP are equivalent definitions!
- WOP proofs involve assuming the opposite, defining $S = \{\text{items where contadiction holds}\}$, dividing m (usually into $m-1$), proving $m-1 \notin S$, and connecting it to $m \in S$ to show $m \notin S$.
- In CSC236, I guess we don't have to rigorously prove the set we're using is a bijection of a subset of $\mathbb{N}$.

Recursively-Defined Set: A set defined by its simplest elements and all possible combinations of them.

Let E be defined as the smallest such that:

- $\forall i \in \mathbb{N}, \quad x_i \in E$
- $\forall e_1, e_2 \in E, \; (e_1 + e_2) \in E$ and $(e_1 \times e_2) \in E$

Structural Induction: Induction based on a recursively-defined set.

Base Case: $P(\text{simplest element})$

Inductive Step: $P(\text{simple elements}) \Rightarrow P(\text{combinations of simple elements})$ (eg. $P(x), P(y) \Rightarrow P(x+y), P(x \times y)$)

Define $P(e)\colon \text{OperatorCount}(e) = \text{VariableCount}(e) - 1$

Show $\forall e \in E, P(e)$

Base Case: Show $\forall i \in \mathbb{N}, P(x_i)\colon \text{OperatorCount}(x_i) = \text{VariableCount}(x_i) - 1$

Let $i \in \mathbb{N}$

$$\text{OperatorCount}(x_i) = 0 = 1 - 1 = \text{VariableCount}(x_i) - 1$$

Inductive Step: Show $\forall e_1, e_2 \in E, P(e_1) \wedge P(e_2) \Rightarrow P\big((e_1 + e_2)\big) \wedge P\big((e_1 \times e_2)\big)$

Let $e_1, e_2 \in E$

Assume $P(e_1)\colon \text{OperatorCount}(e_1) = \text{VariableCount}(e_1) - 1$

Assume $P(e_2)\colon \text{OperatorCount}(e_2) = \text{VariableCount}(e_2) - 1$

Show $P\big((e_1 + e_2)\big)\colon \text{OperatorCount}\big((e_1 + e_2)\big) = \text{VariableCount}\big((e_1 + e_2)\big) - 1$

$$\begin{aligned}\text{OperatorCount}\big((e_1 + e_2)\big) &= \text{OperatorCount}(e_1) + 1 + \text{OperatorCount}(e_2) \\ &= (\text{VariableCount}(e_1) - 1) + 1 + (\text{VariableCount}(e_2) - 1) \\ &= \text{VariableCount}(e_1) + \text{VariableCount}(e_2) - 1 \\ &= \text{VariableCount}\big((e_1 + e_2)\big) - 1\end{aligned}$$

Show $P(e_1 \times e_2)\colon \text{OperatorCount}(e_1 \times e_2) = \text{VariableCount}(e_1 \times e_2) - 1$

By literally the same steps as above, you can show this.

Correctness

Preconditions: A predicate that's true before a function executes. Ideally, weak constraints.

Postconditions: A predicate that's true after a function executes. Ideally, strong constraints.

Loop Invariant (LI): A predicate that's true for every iteration of a loop before that iteration executes

- Must be separately claimed & proven via induction on iteration number
- Might need expressed as LI_k or $\text{LI}(k)$ in terms of k, the iteration number.

Partially Correct: A program where, if preconditions hold and the program terminates, postconditions hold

- ➢ *To Prove It:* Assume preconditions, assume termination, show postconditions

Totally Correct: A partially correct program that terminates.

- ➢ *To Prove It:* Show partial correctness, show termination

Precondition:	$x \in \mathbb{R}, y \in \mathbb{N}$	
Loop Precondition:	$x \in \mathbb{R}, y \in \mathbb{N}, z = 1, m = 0$	(simple explanation)
Loop Invariant:	$z = x^m, m \leq y, m \in \mathbb{N}$	(prove this)
Loop Condition:	$x < y$	
Loop Postcondition:	$z = x^m, m \leq y, m \in \mathbb{N}, m \geq y$	(simple explanation)
Postcondition:	return x^y	(prove this)

```
pow(x, y):
    z = 1
    m = 0
    while m < y:
        z = z * x
        m = m + 1
    return z
```

Proof Techniques:

No recursion/loops	Analyze code line-by-line.
Recursion	Show preconditions hold on recursive call Proceed via induction (even if there are loops inside) Show input size of recursive call is smaller than that of original call Thus postconditions for the recursive call hold
Loops	Prove a LI exists (For nested loops, outer, inner LIs. For consecutive loops, multiple LIs) • Show LI holds before loop executes • Assume LI holds at iteration start, show LI holds before next iteration (via induction) • Assume LI holds at loop end, show LI postconditions hold Prove loop termination (when proving termination) • While Loop: Find a decreasing sequence $E_k \subseteq \mathbb{N}$ where $k =$ iteration count and $E_k =$ upper bound on remaining iterations. By WOP, E_k has a minimum, therefore it is finite, so the loop has finitely many iterations and terminates • For Loop: By definition, for loops terminate. Can be written as a while loop. ○ For a loop from a to b, choose $E = b + 1 - x$ to prove termination (since $b, x \in \mathbb{N}, x \leq b + 1$) ○ Element-based for loops can be written as looping over indices. Thus postconditions for loop hold

Show $\text{pow}(x, y)$ is partially correct

Assume preconditions, $x \in \mathbb{R}, y \in \mathbb{N}$

Assume program terminates

Show postcondition, x^y is returned

Claim LI: $z = x^m \wedge m \leq y \wedge m \in \mathbb{N}$

Show $\forall k \in \mathbb{N}$, LI_k holds

Let $k \in \mathbb{N}$

Base Case: Show LI_0: $z = x^m \wedge m \leq y \wedge m \in \mathbb{N}$

At line 3, before the loop starts, we know $z = 1, m = 0$

$$z = 1 = x^0 = x^m$$

$$m = 0 \in \mathbb{N} \text{ and } 0 < y \text{ (since } y \in \mathbb{N})$$

Inductive Step: Show $\text{LI}_k \Rightarrow \text{LI}_{k+1}$

Assume at least $k+1$ iterations (otherwise, $\text{LI}_{k+1} = \text{LI}_k$ via the IH, a trivial result), therefore loop condition is true, $m_k < y$

Assume LI_k: $z_k = x^{m_k} \wedge m_k \leq y \wedge m \in \mathbb{N}$

Show LI_{k+1}: $z_{k+1} = x^{m_{k+1}} \wedge m_{k+1} \leq y \wedge m + 1 \in \mathbb{N}$

By line 5,	$m_{k+1} = m_k + 1 \in \mathbb{N}$.
Since $m_k < y$, and $m_k, y \in \mathbb{N}$,	$m_k + 1 \leq y$
By line 4,	$z_{k+1} = z_k x$
Since $z_k = x^{m_k}$,	$z_{k+1} = x^{m_k} x = x^{m_k + 1}$

Then by induction, LI_k holds for all $k \in \mathbb{N}$.

Since the program terminates, the loop terminates.

At loop termination, the LI is true and the loop condition is false, so $z = x^m, m \leq y, m \geq y$.

Since $m \leq y$ and $m \geq y$, then $m = y$.

Then $z = x^m = x^y$ is returned, as required by the postcondition.

Show $\text{pow}(x, y)$ terminates

Let $k \in \mathbb{N}$ be the number of iterations of the while loop

Pick $E_k = y - m_k$

Show E is finite

By preconditions, $m, y \in \mathbb{N}$,	we know $E \in \mathbb{Z}$
By LI, which has $m \leq y$,	we know $E = y - m \geq 0$

Case 1: There're at most k iterations

Then E is finite, as E_k is the last value.

Case 2: There're at least $k+1$ iterations

$$\begin{aligned} E_{k+1} &= y - m_{k+1} \\ &= y - (m_k + 1) \\ &= (y - m_k) - 1 \\ &= E_k - 1 \end{aligned}$$

E is decreasing and $E \subseteq \mathbb{N}$, so by PWO, E has a minimum.

Then E is finite.

Since E_k is finite, then there are finitely many iterations, meaning the loop terminates.

The rest of the code terminates trivially.

Then the function as a whole terminates.

- Variables that don't change throughout iterations shouldn't have subscripts.
- You can just say "assume there are $k+1$" iterations in proving partial correctness and termination.
- You can move the LI claim outside of the partial correctness proof for clearness

Precondition:	lst is a sortable list
Outer LI:	$\text{lst}[0{:}\,i]$ is sorted, $\text{lst}[0{:}\,i] \le \text{lst}[i{:}\,]$ (and $i \in \mathbb{N}, 0 \le i \le \text{len}(\text{lst}) - 1$)
Inner LI:	$\text{lst}[m] = \min(\text{lst}[i{:}\,j])$ (and $j \in \mathbb{N}, 1 + i \le j \le \text{len}(\text{lst}) - 1$)
Postcondition:	lst is sorted in nondecreasing order (and contains the same elements)

```
   select_sort(lst: list):
1.     for i in range(0, len(lst) - 1):
2.         m = i
3.         for j in range(i + 1, len(lst) - 1):
4.             if lst[j] < lst[m]:
5.                 m = j
6.         lst[i], lst[m] = lst[m], lst[i]
```

```
Claim ILI: $\text{lst}[m] = \min(\text{lst}[i{:}\,j])$
    Show $\forall k \in \mathbb{N}, \text{ILI}_k$ holds
        Let $k \in \mathbb{N}$
        Base Case: Show $\text{LI}_0$: $\text{lst}[m_0] = \min(\text{lst}[i{:}\,j_0])$
            At line 2, before the loop starts, we know $m_0 = i$.
            At line 3, the first value of $j$ is $j_0 = i + 1$
                $\min(\text{lst}[i{:}\,j_0]) = \min(lst[i{:}\,i+1]) = \min(lst[i]) = \text{lst}[i] = \text{lst}[m_0]$
        Inductive Step: Show $\text{LI}_k \Rightarrow \text{LI}_{k+1}$
            Assume at least $k+1$ iterations (otherwise, $\text{LI}_{k+1} = \text{LI}_k$ via the IH, a trivial result)
            Assume $\text{LI}_k$: $\text{lst}[m_k] = \min(\text{lst}[i{:}\,j_k])$
            Show $\text{LI}_{k+1}$: $\text{lst}[m_{k+1}] = \min(\text{lst}[i{:}\,j_{k+1}])$
                Case 1: $\text{lst}[j_{k+1}] \ge \text{lst}[m_k]$
                    Lines 4-5 don't activate, so $m_{k+1} = m_k$
                    Then $\min(\text{lst}[i{:}\,j_{k+1}]) = \min(\text{lst}[i{:}\,j_k]) = \text{lst}[m_k] = \text{lst}[m_{k+1}]$
                Case 2: $\text{lst}[j_{k+1}] < \text{lst}[m_k]$
                    Lines 4-5 activate, so $m_{k+1} = j_{k+1}$
                    Since $\text{lst}[m_{k+1}] = \text{lst}[j_{k+1}] < \text{lst}[m_k] = \min(\text{lst}[i{:}\,j_k])$,
                    Then $\text{lst}[m_{k+1}] = \min(\text{lst}[i{:}\,j_{k+1}])$
        Then by induction, $\text{LI}_k$ holds for all $k \in \mathbb{N}$.

Claim OLI: $\text{lst}[0{:}\,i]$ is sorted $\wedge$ $\text{lst}[0{:}\,i] \le \text{lst}[i{:}\,]$
    Show $\forall k \in \mathbb{N}, \text{LI}_k$ holds
        Let $k \in \mathbb{N}$
        Base Case: Show $\text{LI}_0$: $\text{lst}[0{:}\,i_0]$ is sorted $\wedge$ $\text{lst}[0{:}\,i_0] \le \text{lst}[i_0{:}\,]$
            The loop starts at $i_0 = 0$, making both statements vacuously true.
        Inductive Step: Show $\text{LI}_k \Rightarrow \text{LI}_{k+1}$
            Assume at least $k+1$ iterations (otherwise, $\text{LI}_{k+1} = \text{LI}_k$ via the IH, a trivial result)
            Assume $\text{LI}_k$: $\text{lst}[0{:}\,i_k]$ is sorted $\wedge$ $\text{lst}[0{:}\,i_k] \le \text{lst}[i_k{:}\,]$
            Show $\text{LI}_{k+1}$: $\text{lst}[0{:}\,i_{k+1}]$ is sorted $\wedge$ $\text{lst}[0{:}\,i_{k+1}] \le \text{lst}[i_{k+1}{:}\,]$
                By line 2, $m_{k+1} = i_{k+1}$
                Since the program terminates, the inner loop terminates at $j = \text{len}(\text{lst}) - 1$
                By ILI,          $\text{lst}[m_{k+1}] = \min(\text{lst}[i_{k+1}{:}\,j]) = \min(\text{lst}[i_{k+1}{:}\,])$
                After line 6,    $\text{lst}[i_{k+1}] = \text{lst}[m_{k+1}]$
                Then             $\text{lst}[i_{k+1}] = \min(\text{lst}[i_{k+1}{:}\,])$, so $\text{lst}[i_{k+1}] \le \text{lst}[i_{k+1}{:}\,]$
                From IH,         $\text{lst}[0{:}\,i_k] \le \text{lst}[i_{k+1}]$ and $\text{lst}[0{:}\,i_k]$ is sorted
                Then $\text{lst}[0{:}\,i_{k+1}]$ is sorted
        Then by induction, $\text{LI}_k$ holds for all $k \in \mathbb{N}$.
```

Show $\text{SelectSort}(\text{lst})$ is partially correct

    Assume precondition, lst is a sortable list
    Assume program terminates
    Show postcondition, lst is sorted in nondecreasing order, elements are the same
        Since the program terminates, both loops terminate. Outer loop terminates at $i = \text{len}(\text{lst}) - 1$.
        Since OLI is true, $\text{lst}[0{:}\,\text{len}(\text{lst}) - 1]$ is sorted $\wedge\ \text{lst}[0{:}\,\text{len}(\text{lst}) - 1] \leq \text{lst}[\text{len}(\text{lst}) - 1{:}\,]$
        Since $\text{lst}[0{:}\,\text{len}(\text{lst}) - 1] \leq \text{lst}[\text{len}(\text{lst}) - 1{:}\,] = \text{lst}[\text{len}(\text{lst}) - 1]$,
        Then $\text{lst}[0{:}\,\text{len}(\text{lst})] = \text{lst}$ is sorted.
        Line 6 is the only mutating operation, and it switches the positions of two list items
        Then list returns all of its original elements.

Show $\text{SelectSort}(\text{lst})$ terminates

    Let $k \in \mathbb{N}$ be the number of iterations of the inner loop
    Pick $E_k = \text{len}(\text{lst}) - 1 - j_k$
    Show E is finite
        (We need the LI that I put in brackets; they're easy to prove, just time-taking and annoying)
        Since $i + 1 \leq j_k \leq \text{len}(\text{lst}) - 1$,   we know $E \geq 0$
        Since $i, j_k \in \mathbb{N}$,   we know $E = \text{len}(\text{lst}) - 1 - j_k \in \mathbb{Z}$
        Case 1: There're at most k iterations
            Then E is finite, as E_k is the last value.
        Case 2: There're at least $k + 1$ iterations

$$\begin{aligned} E_{k+1} &= \text{len}(\text{lst}) - 1 - j_{k+1} \\ &= \text{len}(\text{lst}) - 1 - (j_k + 1) \ \text{(as inner for loop steps by 1)} \\ &= (\text{len}(\text{lst}) - 1 - j_k) - 1 \\ &= E_k - 1 \end{aligned}$$

            E is decreasing and by PWO, E has a minimum.
            Then E is finite.
    Since E_k is finite, then there are finitely many iterations, meaning the loop terminates.
    The same thing can be done to show the outer loop terminates. *(Usually, if you're not specifically told to prove a for loop terminates, you can just say it terminates)*
    The rest of the code terminates trivially.
    Then the function as a whole terminates.

Preconditions: $b, e \in \mathbb{N}$
A's elements comparable with x
$A[b\colon e]$ is sorted
$0 \le b < e \le \text{len}(A)$
Postconditions: Returns $p \in \mathbb{Z}$ such that
$b \le p \le e$
$p > b \Rightarrow A[p-1] < x$
$p < e \Rightarrow A[p] \ge x$

```
    RecBinSearch($x, A, b, e$):
1.      if $e == b + 1$:
2.          if $x \leqslant A[b]$:
3.              return $b$
            else:
4.              return $e$
        else:
5.          $m = \lfloor (b+e)/2 \rfloor$
6.          if $x \leqslant A[m-1]$:
7.              return RecBinSearch($x, A, b, m$)
            else:
8.              return RecBinSearch($x, A, m, e$)
```

Show $\text{RecBinSearch}(x, A, b, e)$ is correct

Let $P(n)$: For all inputs of size $n = e - b$ satisfying preconditions, $\text{RecBinSearch}(x, A, b, e)$ terminates and satisfies postconditions

Show $\forall n \in \mathbb{N}, P(n)$ holds

Let $n \in \mathbb{N}$

Base Case: Show $P(1)$

Assume $n = e - b = 1$, so $e = b + 1$

Assume all input satisfy preconditions

Show $\text{RecBinSearch}(x, A, b, e)$ terminates and satisfies postconditions

Since $e = b + 1$, we pass into the if branch of line 1.

Then either b or e is returned, terminating the program.

Then for $p \in \{b, e\}$, $b \le p \le e$ holds; other 2 postconditions vacuously true.

Inductive Step: Show $(\forall k \in \mathbb{N}, k < n \Rightarrow P(k)) \Rightarrow P(n)$ *(assume $n \ge 2$)*

Assume $\forall k \in \mathbb{N}, k < n \Rightarrow P(k)$

Show $P(n)$

Since $n = e - b \ge 2$, then $e \ne b + 1$, so we pass into the else branch of line 1.

By line 5, $m = \lfloor \frac{b+e}{2} \rfloor$

Since $b < e$, then $m = \lfloor \frac{b+e}{2} \rfloor \le \frac{b+e}{2} < \frac{2e}{2} = e$

Since $e > b$, then $m = \lfloor \frac{b+e}{2} \rfloor > \lfloor \frac{2b}{2} \rfloor = \lfloor b \rfloor = b$

Case 1: $x \le A[m-1]$

The if branch of line 6 activates.

Since $\lfloor \frac{b+e}{2} \rfloor \in \mathbb{N}$, then $m = \lfloor \frac{b+e}{2} \rfloor \in \mathbb{N}$

Since $b < m < e$ and $A[b\colon e]$ is sorted, then $A[b\colon m]$ is sorted

Since $0 \le b < m < e \le \text{len}(A)$, then $0 \le b < m \le \text{len}(A)$

Since $b < m < e$, then $n = e - b > m - b > 0$

Then by IH, for $P(m - b)$, since preconditions are satisfied, then recursive call will terminate and its postconditions will hold.

Recursive Call	*Want to Show*
$b \le p \le m$	$b \le p \le e$
$p > b \Rightarrow A[p-1] < x$	$p > b \Rightarrow A[p-1] < x$
$p < m \Rightarrow A[p] \ge x$	$p < e \Rightarrow A[p] \ge x$

#1 is true as $b \le p \le m < e$, while #2 is true trivially. #3 is also true:

If $p < m$, then conditional holds and $A[p] \ge x$.

If $p = m$, then $x \le A[m-1] \le A[m] = A[p]$ *(as A sorted)*

By #1, $p > m$ is impossible

Then all postconditions are satisfied

Case 2: $x > A[m-1]$

The else branch of line 6 activates.

Since $\lfloor \frac{m+e}{2} \rfloor \in \mathbb{N}$, then $m = \lfloor \frac{b+e}{2} \rfloor \in \mathbb{N}$

Since $b < m < e$ and $A[b\colon e]$ is sorted, then $A[m\colon e]$ is sorted

Since $0 \leq b < m < e \leq \text{len}(A)$, then $0 \leq m < e \leq \text{len}(A)$

Since $b < m < e$, then $n = e - b > e - m > 0$

Then by IH, for $P(e-m)$, since preconditions are satisfied, then recursive call will terminate and its postconditions will hold.

Recursive Call	*Want to Show*
$m \leq p \leq e$	$b \leq p \leq e$
$p > m \Rightarrow A[p-1] < x$	$p > b \Rightarrow A[p-1] < x$
$p < e \Rightarrow A[p] \geq x$	$p < e \Rightarrow A[p] \geq x$

#1 is true as $b < m \leq p \leq e$, while #3 is true trivially. #2 is also true:

If $p > m$, then conditional holds and $A[p-1] < x$.

If $p = m$, then $x > A[m-1] = A[p-1]$

By #1, $p < m$ is impossible

Then all postconditions are satisfied

Running-Time Analysis

Step: A sequence of code that execute in constant time

Running-Time Analysis: Analyzing number of steps as a function of input size

- Focus on worst-case measure, $T(n)$.
- Often, no simple expression for $T(n)$, prove bounds using asymptotic notation

Big-O: A running-time has an upper bound, $T(n) \in \mathcal{O}(f(n)) \Leftrightarrow \exists n_0, c \in \mathbb{R}^+, \forall n \geq n_0, T(n) \leq c \cdot f(n)$

Omega: A running-time has a lower bound, $T(n) \in \Omega(f(n)) \Leftrightarrow \exists n_0, c \in \mathbb{R}^+, \forall n \geq n_0, T(n) \geq c \cdot f(n)$

Theta: A running-time has a tight bound, $T(n) \in \Theta(f(n)) \Leftrightarrow T(n) \in \mathcal{O}(f(n))$ and $T(n) \in \Omega(f(n))$

Master Theorem: For $a, n_0 \in \mathbb{Z}^+, b, k \in \mathbb{R}, b > 1, k \geq 0$, we can solve recurrences relations of the form

$$T(n) = \begin{cases} 1 & n \leq n_0 \\ aT\left(\frac{n}{b}\right) + n^k & n > n_0 \end{cases} = \begin{cases} \Theta(n^k) & a < b^k \ (\log_b a < k) \\ \Theta(n^k \log n) & a = b^k \ (\log_b a = k) \\ \Theta(n^{\log_b a}) & a > b^k \ (\log_b a > k) \end{cases}$$

Requires recursive call input sizes to be roughly $T(\frac{n}{b})$, but we can ignore floors/ceilings and small constants.

Repeated Substitution: Technique to guess a tight bound before formally proving it.

```
FACT(n):
1.   if n ⩽ 1:
2.       return 1
3.   return n × FACT(n − 1)
```

Find worst-case running-time $T(n)$ recursively, where n is the input size.

Base Case: $n \leq 1$

Then line 1 (constant-time) and 2 (constant-time) run. Thus there is 1 step.

Recursive Case: $n > 1$

Then lines 1 (constant-time, 1 step) and 3 run. Then input size for line 3 is $n-1$, so there are $T(n-1)$ steps.

$$\therefore T(n) = \begin{cases} 1 & n = 1 \\ 1 + T(n-1) & n > 1 \end{cases}$$

Find a tight bound for $T(n)$. Use **repeated substitution.**

$$\begin{aligned} T(n) &= 1 + T(n-1) \\ &= 1 + (1 + T(n-2)) = 2 + T(n-2) \\ &= 2 + (1 + T(n-3)) = 3 + T(n-3) \\ &= \cdots \\ &= k + T(n-k) \end{aligned}$$

Make $T(n) = k + T(n-k)$ not a recurrence relation by getting rid of $T(n-k)$ with a value of k. Try $k = n-1$:

$$\begin{aligned} T(n) &= (n-1) + T(n-(n-1)) \\ &= n - 1 + T(1) \\ &= n \end{aligned}$$

We can then formally prove $T(n) = n$ using induction on $n \in \mathbb{N}$.

Let $n \in \mathbb{N}$

Base Case: Show $P(1)$:

$T(1) = 1$ by definition so this is trivial.

Inductive Step: Show $P(n) \Rightarrow P(n+1)$

Assume $T(n) = n$

Show $T(n+1) = n+1$

$$\begin{aligned} T(n+1) &= 1 + T((n+1) - 1) \\ &= T(n) + 1 \\ &= n + 1 \end{aligned}$$

Therefore, $T(n) = n \in \Theta(n)$

Find worst-case running-time $T(n)$ where $n = e - b$ is input size.

Base Case: $T(1)$

Then $n = e - b = 1$, so $e = b + 1$.

Lines 1-4 are constant-time, so 1 step.

$$\therefore T(1) = 1$$

Recursive Case: $T(n)$ for $n > 1$

Lines 1, 5, 6 are constant-time, so 1 step.

Input size for line 7 is $m - b = \lfloor \frac{b+e}{2} \rfloor - b = \lfloor \frac{b+e-2b}{2} \rfloor = \lfloor \frac{e-b}{2} \rfloor = \lfloor \frac{n}{2} \rfloor$

Input size for line 8 is $e - m = e + \lceil -\frac{b+e}{2} \rceil = \lceil \frac{2e-b-e}{2} \rceil = \lceil \frac{e-b}{2} \rceil = \lceil \frac{n}{2} \rceil$

We take the worst-case running-time, the max of these values:

$$\therefore T(n) = 1 + \max\left\{T\left(\left\lfloor \frac{n}{2} \right\rfloor\right), T\left(\left\lceil \frac{n}{2} \right\rceil\right)\right\} \left(= 1 + T\left(\left\lceil \frac{n}{2} \right\rceil\right)\right)$$

```
    RecBinSearch($x, A, b, e$):
1.      if $e == b + 1$:
2.          if $x \leqslant A[b]$:
3.              return $b$
            else:
4.              return $e$
        else:
5.          $m = \lfloor (b + e)/2 \rfloor$
6.          if $x \leqslant A[m - 1]$:
7.              return RecBinSearch($x, A, b, m$)
            else:
8.              return RecBinSearch($x, A, m, e$)
```

First, let's show $\max\{T(\lfloor \frac{n}{2} \rfloor), T(\lceil \frac{n}{2} \rceil)\} = T(\lceil \frac{n}{2} \rceil)$. This is true if T is non-decreasing,

Show T is non-decreasing, meaning $\forall n_0, n_1 \in \mathbb{N}, n_0 < n_1 \Rightarrow T(n_0) \leq T(n_1)$

Let $n_1 \in \mathbb{N}$

Let $P(n_1)$: $\forall n_0 \in \mathbb{N}, n_0 < n_1 \Rightarrow T(n_0) \leq T(n_1)$

Base Cases: $P(1), P(2)$

Show $P(1)$: $\forall n_0 \in \mathbb{N}, n_0 < 1 \Rightarrow T(n_0) \leq T(1)$ Vacuously true; $n_0 \in \mathbb{N}, n_0 < 1$ impossible

Show $P(2)$: $\forall n_0 \in \mathbb{N}, n_0 < 2 \Rightarrow T(n_0) \leq T(2)$

Let $n_0 \in \mathbb{N}$

Assume $n_0 < 2$, so $n_0 = 1$

Show $T(n_0) \leq T(2)$ $\therefore T(1) = 1 \leq 1 + \max\{T(\lfloor \frac{2}{2} \rfloor), T(\lceil \frac{2}{2} \rceil)\} = T(2)$

Inductive Step: Show $\forall k > 2, (\forall k' < k, P(k')) \Rightarrow P(k)$

Let $k > 2$

Assume $\forall k' < k, P(k')$: $\forall n_0 \in \mathbb{N}, n_0 < k' \Rightarrow T(n_0) \leq T(k')$

Show $P(k)$: $\forall n_0 \in \mathbb{N}, n_0 < k \Rightarrow T(n_0) \leq T(k)$

Let $n_0 \in \mathbb{N}$

Assume $n_0 < k$

Show $T(n_0) \leq T(k)$

Since $\lfloor \frac{n_0}{2} \rfloor < \lceil \frac{k}{2} \rceil < k$, by IH, $T(\lfloor \frac{n_0}{2} \rfloor) \leq T(\lceil \frac{k}{2} \rceil)$

$$\begin{aligned} T(n_0) &= 1 + \max\left\{T\left(\left\lfloor \frac{n_0}{2} \right\rfloor\right), T\left(\left\lceil \frac{n_0}{2} \right\rceil\right)\right\} \\ &\leq 1 + \max\left\{T\left(\left\lceil \frac{k}{2} \right\rceil\right), T\left(\left\lceil \frac{k}{2} \right\rceil\right)\right\} \\ &= 1 + T\left(\left\lceil \frac{k}{2} \right\rceil\right) \\ &= T(k) \end{aligned}$$

Now, we simplify the expression and apply repeat substitution.

$$T(n) \approx \begin{cases} 1 & n = 1 \\ 1 + T\left(\frac{n}{2}\right) & n > 1 \end{cases}$$

$$\begin{aligned} T(n) &\approx 1 + T\left(\frac{n}{2}\right) \\ &= 1 + \left(1 + T\left(\frac{n}{4}\right)\right) = 2 + T\left(\frac{n}{4}\right) \\ &= 2 + \left(1 + T\left(\frac{n}{8}\right)\right) = 3 + T\left(\frac{n}{8}\right) \\ &= \cdots \\ &= k + T\left(\frac{n}{2^k}\right) \end{aligned}$$

To remove $T(\frac{n}{2^k})$, we can set $2^k = n$ or $k = \log_2 n$.

$$\begin{aligned} T(n) &\approx k + T\left(\frac{n}{2^k}\right) \\ &= \log_2 n + T(1) \\ &= \log_2 n + 1 \end{aligned}$$

While the answer is not necessarily true, our tight bound is probably $\Theta(\log n)$, which we will now prove formally.

Show $T(n) \in \mathcal{O}(\log_2(n-1)+2)$, meaning $\exists c, n_0 \in \mathbb{R}^+, \forall n \in \mathbb{N}, n \geq n_0 \Rightarrow T(n) \leq \log_2(n-1)+2$

Pick $c = 1$

Pick $n_0 = 2$

Let $n \in \mathbb{N}$

Let $P(n)\colon n \geq 2 \Rightarrow T(n) \leq \log_2(n-1)+2$

Base Case: Show $P(2)$ $\quad \therefore T(2) = 1 + T(1) = 2 = \log_2(2-1)+2$

Inductive Step: Show $\forall k > 2, \big(\forall k' < k, P(k')\big) \Rightarrow P(k)$

Let $k > 2$

Assume $\forall k' < k, P(k')\colon k' \geq 2 \Rightarrow T(k') \leq \log_2(k'-1)+2$

Show $P(k)\colon k \geq 2 \Rightarrow T(k) \leq \log_2(k-1)+2$

$$\begin{aligned}
T(k) &= 1 + T\left(\left\lceil \frac{k}{2} \right\rceil\right) \\
&\leq 1 + \log_2\left(\left\lceil \frac{k}{2} \right\rceil - 1\right) + 2 \\
&\leq 3 + \log_2\left(\frac{k+1}{2} - 1\right) \\
&= 3 + \log_2\left(\frac{k-1}{2}\right) \\
&= 3 + \log_2(k-1) - 1 \\
&= 2 + \log_2(k-1)
\end{aligned}$$

Show $T(n) \in \Omega(\log_2 n)$, meaning $\exists c, n_0 \in \mathbb{R}^+, \forall n \in \mathbb{N}, n \geq n_0 \Rightarrow T(n) \geq \log_2 n$

Pick $c = 1$

Pick $n_0 = 1$

Let $n \in \mathbb{N}$

Let $P(n)\colon n \geq 1 \Rightarrow T(n) \geq \log_2 n$

Base Case: Show $P(1)$ $\quad \therefore T(1) = 1 \geq 0 = \log_2 1$

Inductive Step: Show $\forall k > 1, \big(\forall k' < k, P(k')\big) \Rightarrow P(k)$

Let $k > 1$

Assume $\forall k' < k, P(k')\colon k' \geq 1 \Rightarrow T(k') \geq \log_2 k'$

Show $P(k)\colon k \geq 1 \Rightarrow T(k) \geq \log_2 k$

$$T(k) = 1 + T\left(\left\lceil \frac{k}{2} \right\rceil\right) \geq 1 + \log_2 \left\lceil \frac{k}{2} \right\rceil \geq 1 + \log_2 \frac{k}{2} = 1 + \log_2 k - 1 = \log_2 k$$

We know $T(n) \in \Omega(\log_2 n) = \Omega(\log n)$

We know $T(n) \in \mathcal{O}(\log_2(n-1)+2) = \mathcal{O}(\log n)$

Therefore $T(n) \in \Theta(\log n)$

Find worst-case running-time $T(n)$, $n = \text{len}(A)$ is input size.

Base Case: $T(1)$

Then nothing happens, 1 step.

$$\therefore T(1) = 1$$

Recursive Case: $T(n)$ for $n > 1$

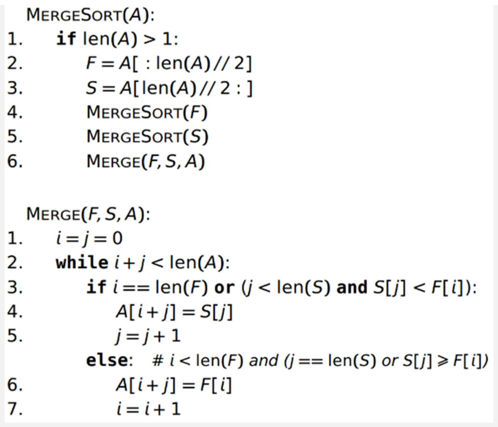
Line 1 is 1 step.

Lines 2, 4 are a recursive call of input size $\lfloor \frac{n}{2} \rfloor$

Lines 3, 5 are a recursive call of input size $\lceil \frac{n}{2} \rceil$

Line 6 is n steps (since in Merge, $k = i + j$ increases by 1 each iteration until $k \geq n$)

$$\therefore T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n + 1$$



```
MERGESORT(A):
1.  if len(A) > 1:
2.      F = A[ : len(A)// 2]
3.      S = A[len(A)// 2 : ]
4.      MERGESORT(F)
5.      MERGESORT(S)
6.      MERGE(F, S, A)

MERGE(F, S, A):
1.  i = j = 0
2.  while i + j < len(A):
3.      if i == len(F) or (j < len(S) and S[j] < F[i]):
4.          A[i + j] = S[j]
5.          j = j + 1
        else:  # i < len(F) and (j == len(S) or S[j] ≥ F[i])
6.          A[i + j] = F[i]
7.          i = i + 1
```

Now, we apply repeated substitution…

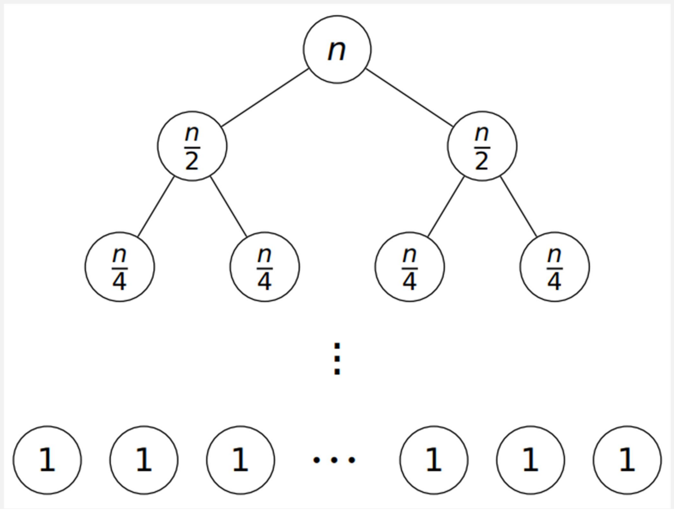
$$T(n) \approx 2T\left(\frac{n}{2}\right) + n + 1$$

$$\begin{aligned}
&= 2\left(2T\left(\frac{n}{2^2}\right)+\frac{n}{2}+1\right)+n+1 = 2^2T\left(\frac{n}{2^2}\right)+2n+(1+2)\\
&= 2^2\left(2T\left(\frac{n}{2^3}\right)+\frac{n}{2^2}+1\right)+2n+(1+2) = 2^3T\left(\frac{n}{2^3}\right)+3n+(1+2+2^2)\\
&= \cdots\\
&= 2^kT\left(\frac{n}{2^k}\right)+kn+\sum_{i=0}^{k-1}2^i
\end{aligned}$$

Set $k = \log_2 n$, then

$$\begin{aligned}
&= nT(1)+n\log_2 n+\sum_{i=0}^{\log_2 n-1}2^i\\
&= n\log_2 n+n+(2^{\log_2 n}-1)\\
&= n\log_2 n+2n-1
\end{aligned}$$

We can alternatively visualize $T(n) = 2T(\frac{n}{2})+n+1$ like:



Nodes	RT/Node	Total RT
1	$n+1$	$n+1$
2	$\frac{n}{2}+1$	$n+2$
4	$\frac{n}{4}+1$	$n+4$
$\vdots$	$\vdots$	$\vdots$
$\frac{n}{2}$	$2+1$	$n+\frac{n}{2}$
n	1	n

Height is $n = 2^h$, or $h = \log_2 n$

Thus RT is $hn+\sum_{i=0}^{h-1}2^i = n\log_2 n+2n-1$

eg. Integer multiplication, $X \times Y$, treat X, Y as lists of base 2 numbers, add 0s in front to equalize list lengths.

Iterative Approach: We multiply each digit of X with each digit of Y, multiply by 10, collect results: $\Theta(n^2)$

Divide-and-Conquer Approach:

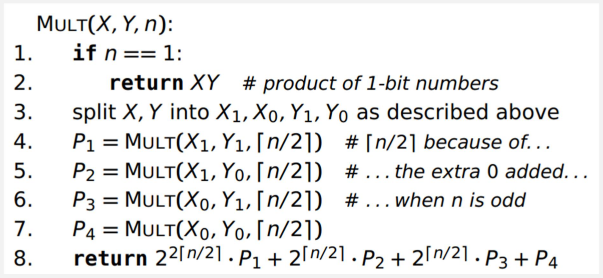
If X, Y are not oddly-lengthed, pad them with a 0 in front.

Bisect X into $X_0 = \left[x_0, \dots, x_{\frac{n}{2}-1}\right], X_1 = \left[x_{\frac{n}{2}}, \dots, x_{n-1}\right]$

Bisect Y into $Y_0 = \left[y_0, \dots, y_{\frac{n}{2}-1}\right], Y_1 = \left[x_{\frac{n}{2}}, \dots, x_{n-1}\right]$

Note that

$$\begin{aligned}
XY &= \left(X_1\left(2^{\frac{n}{2}}\right)+X_0\right)\left(Y_1\left(2^{\frac{n}{2}}\right)+Y_0\right)\\
&= X_1Y_1(2^n)+(X_0Y_1+X_1Y_0)\left(2^{\frac{n}{2}}\right)+X_0Y_0
\end{aligned}$$



```
MULT(X, Y, n):
    if $n == 1$:
        return $XY$    # product of 1-bit numbers
    split $X, Y$ into $X_1, X_0, Y_1, Y_0$ as described above
    $P_1$ = MULT($X_1, Y_1, \lceil n/2 \rceil$)    # $\lceil n/2 \rceil$ because of...
    $P_2$ = MULT($X_1, Y_0, \lceil n/2 \rceil$)    # ...the extra 0 added...
    $P_3$ = MULT($X_0, Y_1, \lceil n/2 \rceil$)    # ...when n is odd
    $P_4$ = MULT($X_0, Y_0, \lceil n/2 \rceil$)
    return $2^{2\lceil n/2 \rceil} \cdot P_1 + 2^{\lceil n/2 \rceil} \cdot P_2 + 2^{\lceil n/2 \rceil} \cdot P_3 + P_4$
```

Running-time of such an algorithm is $T(n) = \begin{cases} 1 & n = 1 \\ 4T(\lceil \frac{n}{2} \rceil)+n & n > 1 \end{cases}$

By Master Theorem, $a = 4, b = 2, k = 1$, since $4 > 2^1$, we have $T(n) \in \Theta(n^{\log_2 4}) = \Theta(n^2)$, no better?

But wait, realize that

$$\begin{aligned}
XY &= (X_0+X_1)(Y_0+Y_1)\\
&= X_0Y_0+X_0Y_1+X_1Y_0+X_1Y_1\\
X_0Y_1+X_1Y_0 &= (X_0+X_1)(Y_0+Y_1)-X_0Y_0-X_1Y_1\\
\therefore XY &= X_1Y_1(2^n)-((X_0+X_1)(Y_0+Y_1)-X_0Y_0-X_1Y_1)\left(2^{\frac{n}{2}}\right)-X_0Y_0
\end{aligned}$$

We have to compute $(X_0+X_1)(Y_0+Y_1)$, but we don't need to find X_0Y_1 and X_1Y_0 anymore.

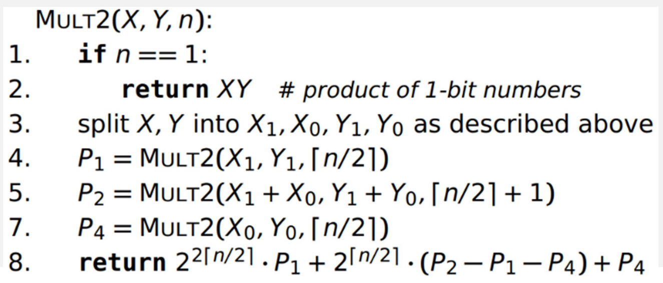
Running-time is now

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lceil \frac{n}{2} \right\rceil + 1\right) + n & n > 1 \end{cases}$$

(I don't know why there's the +1)

Accept that $T(\lceil \frac{n}{2} \rceil + 1) \approx T(\lceil \frac{n}{2} \rceil)$ from the POV of Master Theorem, then $a = 3, b = 2, k = 1$.

Since $4 > 2^1$, then $T(n) \in \Theta(n^{\log_2 3}) \approx \Theta(n^{1.58})$



```
MULT2($X, Y, n$):
    if $n == 1$:
        return $XY$    # product of 1-bit numbers
    split $X, Y$ into $X_1, X_0, Y_1, Y_0$ as described above
    $P_1 = $ MULT2($X_1, Y_1, \lceil n/2 \rceil$)
    $P_2 = $ MULT2($X_1 + X_0, Y_1 + Y_0, \lceil n/2 \rceil + 1$)
    $P_4 = $ MULT2($X_0, Y_0, \lceil n/2 \rceil$)
    return $2^{2\lceil n/2 \rceil} \cdot P_1 + 2^{\lceil n/2 \rceil} \cdot (P_2 - P_1 - P_4) + P_4$
```

Formal Language Theory

Alphabet (Σ): Finite set of symbols

String: Over alphabet Σ, finite sequence of symbols from Σ.

- **Empty String:** The string of length 0, denoted ϵ

Length: Of string s, number of symbols in s, denoted $|s|$

- $\Sigma^n = \{s \text{ over } \Sigma : |s| = n\}$
- $\Sigma^* = \{s \text{ over } \Sigma\} = \bigcup_{i=0}^{\infty} \Sigma^i$

Language (L): Over alphabet Σ, a set $L \subseteq \Sigma^*$.

- $L_1 + L_2 = L_1 \cup L_2$
- $L_1 - L_2 = L_1 \setminus L_2$
- $L_1 \times L_2 = L_1 \cdot L_2 = L_1 L_2$
  $= \{s_1 s_2 \in \Sigma^* : s_1 \in L_1, s_2 \in L_2\}$
- $L^k = \{s_1 \cdots s_k \in \Sigma^* : s_1, .., s_k \in L\}$
- $L^* = \bigcup_{k=0}^{\infty} L^k$ ("Kleene Star")
- $L^+ = \bigcup_{k=1}^{\infty} L^k$
- $\overline{L} = \Sigma^* - L$ ("Complement")

Regular Expression ($\mathcal{R}_\Sigma$, "regex"): Over alphabet Σ, the smallest set containing

- $\emptyset$
- ϵ
- x, for all $x \in \Sigma$
- $(R)^*$, for all $R \in \mathcal{R}_\Sigma$
- $(R_1 R_2)^*$, for all $R_1, R_2 \in \mathcal{R}_\Sigma$
- $(R_1 + R_2)^*$, for all $R_1, R_2 \in \mathcal{R}_\Sigma$

Matched Language ($\mathcal{L}$): A language $\mathcal{L}(\mathcal{R}_\Sigma)$ matched by a regular expression $\mathcal{R}_\Sigma$

- $\mathcal{L}(\emptyset) = \emptyset$
- $\mathcal{L}(\epsilon) = \{\epsilon\}$
- $\mathcal{L}(x) = \{x\}$, for all $x \in \Sigma$
- $\mathcal{L}(R^*) = \left(\mathcal{L}(R)\right)^*$, for all $R \in \mathcal{R}_\Sigma$
- $\mathcal{L}(R_1 R_2) = \mathcal{L}(R_1) \times \mathcal{L}(R_2)$, for all $R_1, R_2 \in \mathcal{R}_\Sigma$
- $\mathcal{L}(R_1 + R_2) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$, for all $R_1, R_2 \in \mathcal{R}_\Sigma$

eg. Prove $b^*a(a+b)^* \equiv (a+b)^*ab^*$

Show $\mathcal{L}(b^*a(a+b)^*) \subseteq \mathcal{L}((a+b)^*ab^*)$

Let $s \in \mathcal{L}(b^*a(a+b)^*)$

Thus $s = s_1 \cdot s_2 \cdot s_3$ for some $s_1 \in \mathcal{L}(b^*), s_2 \in \mathcal{L}(a), s_3 \in \mathcal{L}((a+b)^*)$

Thus $s = b^k \cdot a \cdot u$ for some $k \in \mathbb{N}, u \in \{a,b\}^*$

Case 1: u contains a

Thus u contains a last a, so $u = u' \cdot a \cdot b^l$ for some $u' \in \{a,b\}^*, l \in \mathbb{N}$

Thus $s = b^k \cdot a \cdot (u' \cdot a \cdot b^l)$

We know $b^k \cdot a \cdot u' \in \mathcal{L}((a+b)^*), a \in \mathcal{L}(a), b^l \in \mathcal{L}(b^*)$

Then $s \in \mathcal{L}((a+b)^*ab^*)$

Case 2: u has no a

Thus $s = b^k \cdot a \cdot b^l$ for some $k, l \in \mathbb{N}$

We know $b^k \in \mathcal{L}((a+b)^*), a \in \mathcal{L}(a), b^l \in \mathcal{L}(b^*)$

Then $s \in \mathcal{L}((a+b)^*ab^*)$

Show $\mathcal{L}((a+b)^*ab^*) \subseteq \mathcal{L}(b^*a(a+b)^*)$

Let $s \in \mathcal{L}((a+b)^*ab^*)$

Thus $s = s_1 \cdot s_2 \cdot s_3$ for some $s_1 \in \mathcal{L}((a+b)^*), s_2 \in \mathcal{L}(a), s_3 \in \mathcal{L}(b^*)$

Thus $s = u \cdot a \cdot b^k$ for some $k \in \mathbb{N}, u \in \{a,b\}^*$

Case 1: u contains a

Thus u contains a first a, so $u = b^l \cdot a \cdot u'$ for some $u' \in \{a,b\}^*, l \in \mathbb{N}$

Thus $s = (b^l \cdot a \cdot u') \cdot a \cdot b^k$

We know $b^l \in \mathcal{L}(b^*), a \in \mathcal{L}(a), u' \cdot a \cdot b^k \in \mathcal{L}((a+b)^*)$

Then $s \in \mathcal{L}(b^*a(a+b)^*)$

Case 2: u has no a

Thus $s = b^l \cdot a \cdot b^k$ for some $k, l \in \mathbb{N}$

We know $b^l \in \mathcal{L}(b^*), a \in \mathcal{L}(a), b^k \in \mathcal{L}((a+b)^*)$

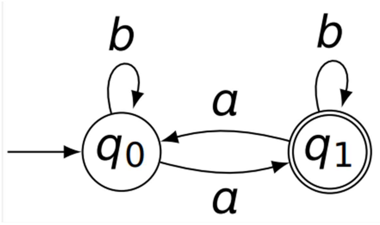
Then $s \in \mathcal{L}(b^*a(a+b)^*)$

Deterministic Finite-State Automaton (DFSA): A flow-chart of "states". Formally, a tuple $\mathcal{D} = (Q, \Sigma, \delta, s, F)$

- Q is a set of all states in $\mathcal{D}$
- Σ is the alphabet of symbols used by $\mathcal{D}$
- $\delta: Q \times \Sigma \to Q$ where is a transition function between states
  - $\delta(q_1, x) = q_2$ means start with state q_1, after processing x, move to state q_2
  - $\delta^*(q_1, x) = q_2$ means do $\delta(q_1, x)$ for every character of x one-by-one

$$\delta^*(q,x) = \begin{cases} q & \text{if } x = \epsilon \\ \delta(\delta^*(q,x_1),x_2) & \text{if } x = x_1x_2 \text{ for some } x_1 \in \Sigma^*, x_2 \in \Sigma \end{cases}$$

- $s \in Q$ is the initial/start state
- $F \subseteq Q$ is a set of accepting/final states



eg. DFSA on the left.

$$\begin{aligned} Q &= \{q_0, q_1\} \\ \Sigma &= \{a, b\} \\ s &= q_0 \\ F &= \{q_1\} \end{aligned} \qquad \begin{aligned} \delta(q_0, a) &= q_1 \\ \delta(q_0, b) &= q_0 \\ \delta(q_1, a) &= q_0 \\ \delta(q_1, b) &= q_1 \end{aligned} \qquad \begin{aligned} \delta^*(q_1, ab) &= \delta(\delta^*(q_1, a), b) \\ &= \delta(\delta(\delta^*(q_1, \epsilon), a), b) \\ &= \delta(\delta(q_1, a), b) \\ &= \delta(q_0, b) \\ &= q_0 \end{aligned}$$

*In diagrams, we omit **dead states**, "dead end" states that can't reach an accepting state

*In diagrams, if $\delta(q, a) = \delta(q, b)$, we use one arrow with a, b instead of two arrows.

Accepts: For a DFSA $\mathcal{D}$, string s if $\delta^*(s, x) \in F$

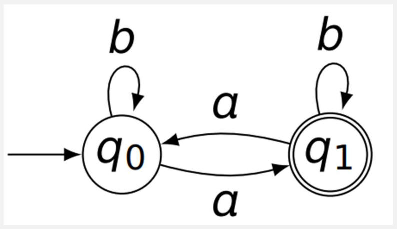
Rejects: For a DFSA $\mathcal{D}$, string s if $\delta^*(s, x) \notin F$

Language: Accepted/recognized by a DFSA $\mathcal{D}$, the language $\mathcal{L}(\mathcal{D}) = \{x \in \Sigma^*: \delta^*(s, x) \in F\}$

State Invariant: Predicate for a state, $P_q(x): \delta^*(s, x) = q$

- To prove state invariants, use a variant of induction
  - Show $P_s(\epsilon)$
  - Show $P_q(x) \Rightarrow P_q(xx')$ for all $x' \in \Sigma$

eg. Show for the DFSA above, $\mathcal{L}(\mathcal{D}) = \{x: x \text{ has odd } a'\text{s}\}$.



Claim $P_{q_0}(x): \delta^*(q_0, x) = \begin{cases} q_0 & x \text{ has even } a'\text{s} \\ q_1 & x \text{ has odd } a'\text{s} \end{cases}$

Basis: $P_s(\epsilon)$

ϵ has 0 (even) a's,

By definition, $\delta^*(s, \epsilon) = s = q_0$.

Recursive Case: $P_{q_0}(x) \Rightarrow P_{q_0}(xx')$ for all $x' \in \Sigma$

Assume $P_{q_0}(x)$

Let $x' \in \Sigma$

Show $P_{q_0}(xx')$

Case 1: $x' = a$

$$\begin{aligned} \delta^*(q_0, xa) &= \delta(\delta^*(q_0, x), a) \\ &= \begin{cases} \delta(q_0, a) & x \text{ has even } a'\text{s} \\ \delta(q_1, a) & x \text{ has odd } a'\text{s} \end{cases} \quad \text{(from IH)} \\ &= \begin{cases} \delta(q_0, a) & xa \text{ has odd } a'\text{s} \\ \delta(q_1, a) & xa \text{ has even } a'\text{s} \end{cases} \\ &= \begin{cases} q_1 & xa \text{ has odd } a'\text{s} \\ q_0 & xa \text{ has even } a'\text{s} \end{cases} \quad (\text{from } \delta \text{ def.}) \end{aligned}$$

Case 2: $x' = b$

$$\begin{aligned} \delta^*(q_0, xb) &= \delta(\delta^*(q_0, x), b) \\ &= \begin{cases} \delta(q_0, b) & x \text{ has even } a'\text{s} \\ \delta(q_1, b) & x \text{ has odd } a'\text{s} \end{cases} \quad \text{(from IH)} \\ &= \begin{cases} \delta(q_0, b) & xb \text{ has even } a'\text{s} \\ \delta(q_1, b) & xb \text{ has odd } a'\text{s} \end{cases} \\ &= \begin{cases} q_0 & xb \text{ has even } a'\text{s} \\ q_1 & xb \text{ has odd } a'\text{s} \end{cases} \quad (\text{from } \delta \text{ def.}) \end{aligned}$$

Show $\mathcal{L}(\mathcal{D}) \subseteq \{x : x \text{ has odd } a'\text{s}\}$

Let $x \in \mathcal{L}(\mathcal{D})$

Since $F = \{q_1\}$, then $\delta^*(s, x) = \delta^*(q_0, x) = q_1$

Recall the state invariant, $\delta^*(q_0, x) = \begin{cases} q_0 & x \text{ has even } a'\text{s} \\ q_1 & x \text{ has odd } a'\text{s} \end{cases}$

Since $\delta^*(q_0, x) = q_1$, then x has odd number of a's

Then $x \in \{x : x \text{ has odd } a'\text{s}\}$

Show $\{x : x \text{ has odd } a'\text{s}\} \subseteq \mathcal{L}(\mathcal{D})$

Let $x \in \{x : x \text{ has odd } a'\text{s}\}$, so x has odd a's

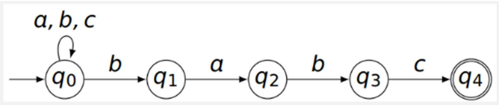
By state invariant, $\delta^*(q_0, x) = q_1$, and $q_1 \in F$

Then $x \in \mathcal{L}(\mathcal{D})$

Non-Deterministic Finite-State Automaton (NFSA): A DSFA that redefines $\delta: Q \times \Sigma \to 2^Q$ (all subsets of Q)

- In other words, $\delta(q, x)$ can have multiple results; NSFAs can be in any number of states simultaneously
- NSFAs accept if some choice of transitions leads to an accepting state

eg. The NSFA for $\{x \in \{a, b, c\}^* : x \text{ ends with } babc\}$



$x = ababa$

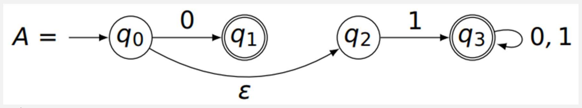
$$\begin{tikzcd}
q_0 \arrow[r, "a"] & q_0 \arrow[r, ")"] \arrow[dr, ")"] & q_1 \arrow[r, "\iota"] & q_2 \arrow[r, ")"] & q_3 \arrow[r, "\iota"] & \text{N/A} & \text{✗} \\
 & & q_0 \arrow[r, "\iota"] & q_0 \arrow[r, ")"] \arrow[dr, ")"] & q_1 \arrow[r, "\iota"] & q_2 \notin F & \text{✗} \\
 & & & & q_0 \arrow[r, "\iota"] & q_0 \notin F & \text{✗}
\end{tikzcd}$$

Then $x = ababa \notin \mathcal{L}(\mathcal{D})$ is rejected by the NFSA.
If any paths lead to something in F, the string is accepted.

ϵ-Transition: Transitions of the form $\delta(q, \epsilon)$, allowing multiple states without a new symbol

- ϵ is notational, don't treat it like an empty string (eg. do not do $0 \cdot 1 \cdot 1 = 0 \cdot \epsilon \cdot 1 \cdot 1$)

eg. The NSFA for $\{x \in \{0,1\}^* : x = 0 \text{ or } x \text{ starts with } 1\}$



$x = 011$

$$\begin{tikzcd}
q_0 \arrow[dr, "\epsilon"] \arrow[rr, ")"] & & q_1 \arrow[r, "-"] & \text{N/A} \arrow[r, "-"] & \text{N/A} & \text{✗} \\
 & q_2 \arrow[r, ")"] & \text{N/A} \arrow[r, "-"] & \text{N/A} \arrow[r, "-"] & \text{N/A} & \text{✗}
\end{tikzcd}$$

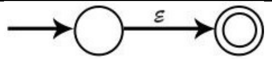
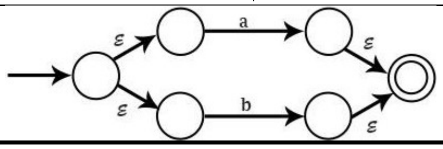
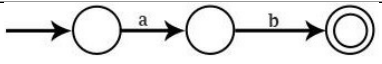
Then $x = 011 \notin \mathcal{L}(\mathcal{D})$ is rejected by the NFSA.

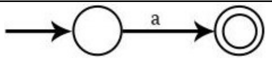
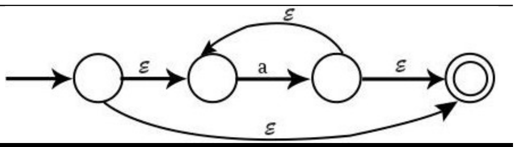
$x = 110$

$$\begin{tikzcd}
q_0 \arrow[dr, "\epsilon"] \arrow[rr, "-"] & & \text{N/A} \arrow[r, "-"] & \text{N/A} \arrow[r, ")"] & \text{N/A} & \text{✗} \\
 & q_2 \arrow[r, "-"] & q_3 \arrow[r, "-"] & q_3 \arrow[r, ")"] & q_3 \in F & \text{✓}
\end{tikzcd}$$

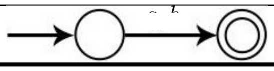
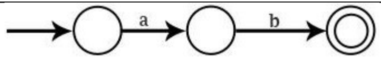
Then $x = 110 \in \mathcal{L}(\mathcal{D})$ is accepted by the NFSA.

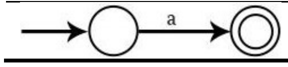
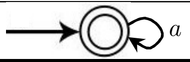
Converting Regex to NFSA

ϵ	$a+b$	ab
		

a	a^*
	

Converting Regex to DFSA

$a+b$	ab
	

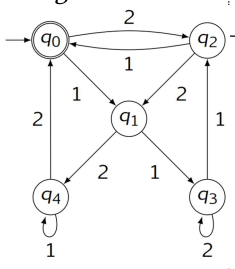
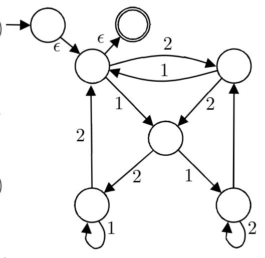
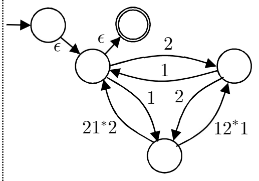
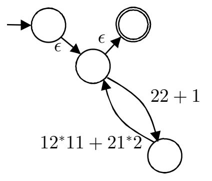
a	a^*
	

Converting DFSA to RE

Add ϵ transitions and modify the DFSA such that:

1) Nothing points to the initial state
2) There is 1 accepting state
3) The accepting state does not point anywhere

Remove states one-by-one, turning them into regex. The Regex to DFSA table can also help here

Original	*Modified*	*Removed Bottom Two*	*Removed Top-Right*	You can simplify this further, but I can already deduce the regex from here:
 q_0 q_2 q_1 q_4 q_3 2 1 1 2 2 1 2 1 1 2				$\big((22+1)(12^*11+21^*2)\big)^*$ You can test this out on the original DFSA to verify it.

Regular: Language L, if (three equivalent definitions)

- $L = \mathcal{L}(\mathcal{D})$ for some DFSA $\mathcal{D}$
- $L = \mathcal{L}(\mathcal{D})$ for some NFSA $\mathcal{D}$
- $L = \mathcal{L}(\mathcal{R}_\Sigma)$ for some regex $\mathcal{R}_\Sigma$

Regular languages over alphabet Σ include:

- $\emptyset$
- $\{\epsilon\}$
- $\{x\}$, for any $x \in \Sigma$
- $L_1 \cup L_2, L_1L_2, L^*, \overline{L}$ for regular languages L_1, L_2

Closed: An operation $\star$ such that if L_1, L_2 are regular, then $L_1 \star L_2$ is also regular

Closure: Property of set to be closed under certain operations (eg. intersection, Kleene star, prefix, reversal)

- $L_1 \cap L_2, L_1 \cup L_2, L_1 \setminus L_2, L_1 \times L_2, L^*, \overline{L}$

eg. Let $L \subseteq \{0,1,2\}^*$ be regular, let $L' = \{x \in \{0,1,2\}^* : x = 1x'$ for some $x' \in L$ or $x = 0x'$ for some $x' \in \overline{L}\}$. Show L' is regular.

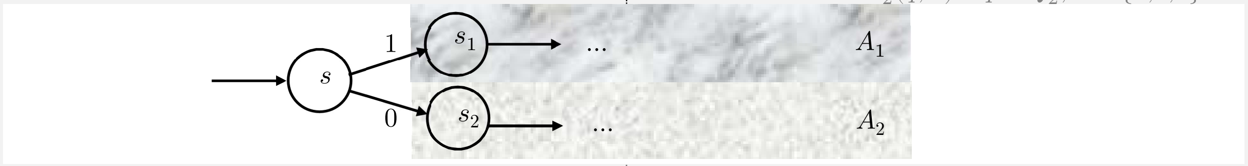
Method 1: DFSAs

Since L is regular, $\exists \mathcal{D}_1 = (Q_1, \{0,1,2\}, \delta_1, s_1, F_1), \mathcal{L}(\mathcal{D}_1) = L$

Since L is regular, $\overline{L}$ is regular, so $\exists \mathcal{D}_2 = (Q_2, \{0,1,2\}, \delta_2, s_2, F_2), \mathcal{L}(\mathcal{D}_2) = \overline{L}$

Define the DFA $\mathcal{D} = (Q, \{0,1,2\}, \delta, s, F)$ such that

- $Q = \{s\} \cup Q_1 \cup Q_2$
- $F = F_1 \cup F_2$
- $\delta(q,x) = \begin{cases} s_1 & q = s, x = 1 \\ s_2 & q = s, x = 0 \\ \delta_1(q,x) & q \in Q_1, x \in \{0,1,2\} \\ \delta_2(q,x) & q \in Q_2, x \in \{0,1,2\} \end{cases}$



If $x = 1x'$ where $x' \in L$, then
$\delta^*(s, 1x') = \delta^*(\delta(s,1), x') = \delta^*(s_1, x') = \delta_1^*(s_1, x')$
Since $x' \in L$, x' is accepted by $\mathcal{D}_1$,
Then $\delta_1^*(s_1, x')$ will return an accepting state.

If $x = 0x'$ where $x' \in \overline{L}$, then
$\delta^*(s, 0x') = \delta^*(\delta(s,0), x') = \delta^*(s_2, x') = \delta_2^*(s_2, x')$
Since $x' \in \overline{L}$, x' is accepted by $\mathcal{D}_2$,
Then $\delta_2^*(s_2, x')$ will return an accepting state.

$\therefore \mathcal{L}(\mathcal{D}) = L'$

Method 2: Regexes

Since L is regular, $\exists R_1$ over $\{0,1,2\}, \mathcal{L}(R_1) = L$
Since L is regular, $\overline{L}$ is regular, so $\exists R_2$ over $\{0,1,2\}, \mathcal{L}(R_2) = \overline{L}$
Define $R = 1R_1 + 0R_2$ over $\{0,1,2\}$, so…

$$\begin{aligned} \therefore \mathcal{L}(R) &= \mathcal{L}(1R_1) \cup \mathcal{L}(0R_2) \\ &= (1 \cdot \mathcal{L}(R_1)) \cup (0 \cdot \mathcal{L}(R_2)) \\ &= (1 \cdot L) \cup (0 \cdot \overline{L}) \\ &= L' \end{aligned}$$

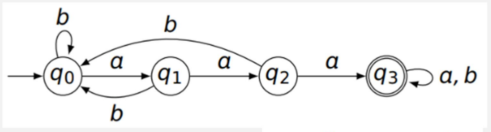
eg. Find DFSA for $L_1 \cap L_2$ with $L_1 = \{x \in \{a,b\}^* : x$ contains $aaa\}$, $L_2 = \{x \in \{a,b\}^* : x$ contains even b's$\}$

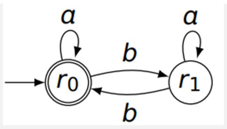
Consider $x = babaa$ for the two DFSAs $\mathcal{D}_1, \mathcal{D}_2$ on the right

$q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2$ ✗ $babaa \notin L_1$

$r_0 \xrightarrow{b} r_1 \xrightarrow{a} r_1 \xrightarrow{b} r_0 \xrightarrow{a} r_0 \xrightarrow{a} r_0$ ✓ $babaa \in L_2$

Therefore $babaa \notin L_1 \cap L_2$



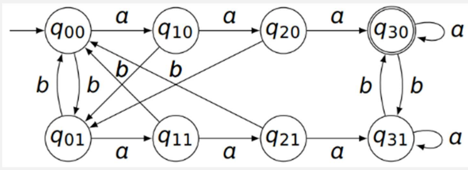


Consider the DFA $\mathcal{D} = (Q, \Sigma, \delta, s, F)$ with

- $Q = Q_1 \times Q_2$
- $\Sigma = \{a,b\}$
- $\delta : Q \times \Sigma \to Q$ where $\delta((q,r),x) = (\delta_1(q,x), \delta_2(r,x))$
- $s \in (s_1, s_2) = (q_0, r_0)$
- $F = F_1 \cap F_2 = (q_3, r_0)$

Show $\mathcal{L}(\mathcal{D}) = L_1 \cap L_2$

$\delta_1^*(q,x) = q_3 \Leftrightarrow x$ contains $aaa \Leftrightarrow x \in L_1$ (to be proved)
$\delta_2^*(r,x) = r_0 \Leftrightarrow x$ contains even b's $\Leftrightarrow x \in L_2$ (to be proved)
$\therefore \delta^*((q,r),x) = (q_3, r_0) \Leftrightarrow x \in L_1 \cap L_2$
$\therefore \mathcal{L}(\mathcal{D}) = L_1 \cap L_2$



Pumping Lemma: For all regular languages $L \subseteq \Sigma^*, \exists p \in \mathbb{Z}^+, \forall x \in L$ with $|x| \geq p, \exists i, j, k \in \Sigma^*$,

- $x = ijk$
- $|ij| \leq p$
- $|j| \geq 1$
- $ij^n k \in L$ for all $n \in \mathbb{N}$

eg. Show $L = \{a^n b^n : n \in \mathbb{N}\} = \{ab, aabb, aaabbb, aaaabbbb, \dots\}$ is not regular.

Method 1: Proof by contradiction
Suppose L is regular, then $\exists \mathcal{D} = (Q, \Sigma, \delta, s, F)$ where $\mathcal{L}(\mathcal{D}) = L$
Consider $x = a^{|Q|+1} b^{|Q|+1} \in L$.
As $|x| > |Q|$, then the path for processing $a^{|Q|+1}$ has a loop passing some state q twice.
Thus $x = a^i a^j a^k b^{|Q|+1}$, where $\delta^*(s, a^i) = q = \delta^*(q, a^j)$
Thus $\delta^*(s, a^i a^k b^{|Q|+1}) = \delta^*(s, a^i a^j a^k b^{|Q|+1})$ even though $a^i a^k b^{|Q|+1} \notin L$, $a^i a^j a^k b^{|Q|+1} \in L$, a contradiction.
Therefore, L is not regular.

Method 2: Proof by contradiction, pumping lemma
Suppose L is regular
Show pumping lemma is false, $\forall p \in \mathbb{Z}^+, \exists x \in L, |x| \geq p, \forall i, j, k \in \Sigma^*, x \neq ijk \vee |ij| > p \vee |j| < 1 \vee ij^n k \notin L$
Let $p \in \mathbb{Z}^+$
Pick $x = a^p b^p \in L$, then $|x| = 2p > p$
Let $i, j, k \in \Sigma^*$
Assume $x = ijk, |ij| \leq p, |j| \geq 1$ *(ie. assume first three premises false, show last premise true)*
Since $x = a^p b^p$ and $|ij| \leq p$, then $i = a^{|i|}, j = a^{|j|}$, meaning $k = a^{p-|ij|} b^p$
Therefore for $n = 1$, $ijk = a^{|i|} a^{|j|} a^{p-|ij|} b^p = a^p b^p = x \in L$, a contradiction.