Q1)

a. $M = (\{p, q\}, \{a, b, c\}, \{S, X, a, b, c\}, \Delta, p, \{q\})$ where

$\Delta = \{$
$((p, a, e), (p, a))$,
$((p, b, e), (p, b))$,
$((p, c, e), (p, c))$,
$((p, e, XaXSa), (p, S))$,
$((p, e, XbXSb), (p, S))$,
$((p, e, c), (p, S))$,
$((p, e, Xa), (p, X))$,
$((p, e, Xb), (p, X))$,
$((p, e, e), (p, X))$,
$((p, e, S), (q, e))$
$\}$

terminals

rules RHS reversed due to stack

start symbol pop → accept (empty stack)

Bottom-up parser mimicks a rightmost derivation of a string $w \in L(G)$ by the CFG G. $(S \overset{*}{\underset{R}{\Rightarrow}}_G w)$.

b. $(p, abbcbabbaa, e) \vdash_m (p, bbcbabbaa, a)$
$\vdash_m (p, bcbabbaa, ba)$
$\vdash_m (p, cbabbaa, bba)$
$\vdash_m (p, babbaa, cbba)$ — c for S
$\vdash_m (p, babbaa, Sbba)$
$\vdash_m (p, babbaa, XSbba)$ — to have XbXSb
$\vdash_m (p, abbaa, bXSbba)$
$\vdash_m (p, abbaa, \underline{XbXSbba})$
$\vdash_m (p, abbaa, Sba)$
$\vdash_m (p, bbaa, aSba)$

%

$\vdash_m (p, bbaa, XaSba)$ ⊤ Xa consumed ⊥

$\vdash_m (p, bbaa, XSba)$

$\vdash_m (p, baa, bXSba)$ ⊤ to have XbXSb ⊥

$\vdash_m (p, baa, XbXSba)$

$\vdash_m (p, baa, Sa)$

$\vdash_m (p, aa, bSa)$

$\vdash_m (p, aa, XbSa)$ ⊤ Xb consumed ⊥

$\vdash_m (p, aa, XSa)$

$\vdash_m (p, a, aXSa)$ ⊤ to have XaXSa Xa consumed ⊥

$\vdash_m (p, a, XSa)$

$\vdash_m (p, e, aXSa)$

$\vdash_m (p, e, XaXSa)$

$\vdash_m (p, e, S)$

$\vdash_m (q, e, e)$ ∎

Q3)

→ cannot go 'back' explicitly

→ no op^{n} to backtrack not like in case of stack which is the most basic element of memory covered in this course

→ only consumes input w/o going over twice the same cell

MEMORYLESS + still has finite control logic ∴ ⇓

acts like a FA (has equivalent computational power) ⇕

regular languages.

Q4)

a. A queue-based deterministic TM M is a quintuple $(K, \Sigma, \delta, s, H)$ where

K is the set of final states,

Σ is the input alphabet containing $\triangleright$ to denote the left end of the queue and not necessarily containing $\sqcup$,

$s \in K$ is the start state,

$H \subseteq K$ is the set of halting states, and

δ is the transition function on

$$(K-H) \times \underbrace{(\Sigma \cup \{e\})}_{\text{front}} \times \underbrace{(\Sigma \cup \{e\})}_{\text{rear}} \to K \times (\Sigma \cup \{\downarrow, \leftrightarrow\})$$

(enqueue, dequeue, no change)

s.t. in $\delta(q, a, b)$ where $q \in K-H$ and $a, b \in \Sigma$

either $\delta(q, e, e)$ or for every a and b either $\delta(q, a, e)$ or $\delta(q, e, b)$ is defined. Otherwise for every b $\delta(q, a, b)$ and for every a $\delta(q, a, b)$ should be defined $\Rightarrow$ deterministic

and for all $q \in K-H$

if $\delta(q, \triangleright, e) = (p, A)$ then $A \neq \downarrow$

$\delta(q, e, \triangleright) = (p, A)$ then $A \neq \downarrow$

} No dequeue on empty queue (only enqueue/no change)

$\delta(q, a, b) = (p, A)$ then $A \neq \triangleright$ → you cannot write $\triangleright$.

b. A configuration of M is a member of

$$K \times \triangleright (\underbrace{(\Sigma - \{\triangleright\}) \cup e}_{\text{front}}) \times \underbrace{(\Sigma^* - \{\triangleright\})}_{\text{rest of the queue content s.t. last character points to rear unless it is } e.}$$

↓ current state

Alternatively, let '$\overline{\ }$' denote front pos^n,
'$\underline{\ }$' denote rear pos^n,

then a config. of M is a member of

$$C \equiv K \times \Delta'(e \cup (\Sigma'(\Sigma^*)(\Sigma' \cup e)))$$

where $\Sigma' = \{\underline{a} : a \in \Sigma\} \cup \{\overline{a} : a \in \Sigma\} \cup \{\underline{\overline{a}} : a \in \Sigma\}$,
$\Delta' = \{\triangleright, \underline{\triangleright}, \overline{\underline{\triangleright}}\}$.

e.g. $(q_1, \overline{\underline{\triangleright}})$ empty queue
$(q_1, \triangleright \underline{\overline{a}})$ queue with one element
$(q_1, \triangleright \overline{a} b \underline{c})$ queue with three elements

c. $\vdash_m \subseteq C \times C$ s.t.
$q_1, q_2 \in K$; $q_1 \in K - H$; $a, b, v, w \in \Sigma - \{\triangleright\}$, $w \in (\Sigma - \{\triangleright\})^*$

$(q_1, \overline{\underline{\triangleright}}) \vdash_m (q_2, \overline{\underline{\triangleright}})$ iff $\delta(q_1, \triangleright, \triangleright) = (q_2, \leftrightarrow)$ (and/or e-cases)

$(q_1, \overline{\underline{\triangleright}}) \vdash_m (q_2, \triangleright \underline{\overline{a}})$ iff $\delta(q_1, \triangleright, \triangleright) = (q_2, a)$ (" ")

$(q_1, \triangleright \underline{\overline{a}}) \vdash_m (q_2, \overline{\underline{\triangleright}})$ iff $\delta(q_1, a, a) = (q_2, \downarrow)$ (" ")

$(q_1, \triangleright \overline{a} w \underline{b}) \vdash_m (q_2, \triangleright \overline{a} w \underline{b})$ iff $\delta(q_1, a, b) = (q_2, \leftrightarrow)$ (" ")

$(q_1, \triangleright \overline{a} \underline{b}) \vdash_m (q_2, \triangleright \underline{\overline{b}})$ iff $\delta(q_1, a, b) = (q_2, \downarrow)$ (" ")

$(q_1, \triangleright \overline{a} v w \underline{b}) \vdash (q_2, \triangleright \overline{v} w \underline{b})$ iff $\delta(q_1, a, b) = (q_2, \downarrow)$ (" ")

$(q_1, \triangleright \overline{a} w \underline{b}) \vdash (q_2, \triangleright \overline{a} w b \underline{c})$ iff $\delta(q_1, a, b) = (q_2, c)$ (" ")

$$L(M) = \{ w \in (\Sigma - \{\triangleright\})^* : (s, \triangleright \overline{w_1} \cdots \underline{w_n}) \vdash_m^* (h, \triangleright \overline{u_1} \cdots \underline{u_m}) \}$$

where $h \in H$, $w = w_1 \cdots w_n$ with $n \in \mathbb{N}$,
$u = u_1 \cdots u_m$ with $m \in \mathbb{N}$, $u \in (\Sigma - \{\triangleright\})^*$,
$u_1, \ldots, u_m, w_1 \cdots w_n \in \Sigma - \{\triangleright\}$ }.

(in finitely many steps $\vdash^*$, $\vdash^N$).

Note that once a halting state is reached queue content at the end should not be part of accepting configuration, as we also want to compute functions using M.

For deciding a language $H = \{y, n\}$ should come into play...

d.(1) Queue-based deterministic TM can do anything the standard TM does.

In standard TM, a snapshot of execution

$$\triangleright\ \alpha_1\ \alpha_2\ \cdots\ \alpha_m\ \underline{a}\ \beta_1\ \beta_2 \cdots \beta_n\ \sqcup\sqcup\cdots$$

- $\alpha_1 \alpha_2 \cdots \alpha_m$: content to the left of the head
- $\underline{a}$: head
- $\beta_1 \beta_2 \cdots \beta_n$: content to the right of the head

represent this snapshot in M as

$$\triangleright\ \overline{a}\ \beta_1\ \beta_2 \cdots \beta_n\ \$\ \alpha_1\ \alpha_2 \cdots \underline{\alpha_m}$$

- $\beta_1 \beta_2 \cdots \beta_n$: content to the right of the head
- $\$$: new separator symbol (start of input tape)
- $\alpha_1 \alpha_2 \cdots \alpha_m$: content to the left of the head

To write a symbol b to where head points to

enqueue end-of-queue symbol '#'

enqueue b

dequeue

$\triangleright\ \overline{a}\ \beta_1 \cdots \beta_n\ \$\ \alpha_1 \cdots \alpha_m\ \underline{\#}$

$\triangleright\ \overline{a}\ \beta_1 \cdots \beta_n\ \$\ \alpha_1 \cdots \alpha_m\ \#\ \underline{b}$

read the front, dequeue enqueue the same symbol till # is at the front

dequeue

$\triangleright\ \overline{\beta_1} \cdots \beta_n\ \$\ \alpha_1 \cdots \alpha_m\ \#\ \underline{b}$

(...)

$\triangleright\ \overline{\#}\ b\ \beta_1 \cdots \beta_n\ \$\ \alpha_1 \cdots \alpha_m$

$\triangleright\ \overline{b}\ \beta_1 \cdots \beta_n\ \$\ \alpha_1 \cdots \alpha_m$

To go right

read front

dequeue

enqueue the read symbol

$\triangleright \overline{\beta_1} \cdots \beta_n \$ \alpha_1 \cdots \alpha_m \underline{a}$

(corresponds to $\triangleright \$ \alpha_1 \cdots \alpha_m a \underline{\beta_1} \cdots \beta_n \sqcup \cdots$ in standard TM)

To go left

mark the end of queue by enqueuing #

$\triangleright \overline{a} \beta_1 \cdots \beta_n \$ \alpha_1 \cdots \alpha_m \underline{\#}$

use finite-state logic (could be large number of states but guaranteed to be finite as Σ is finite) to detect the character α_m that comes before #

read front, dequeue, enqueue the read symbol until α_m

$\triangleright \overline{\alpha_m} \# a \beta_1 \cdots \beta_n \$ \alpha_1 \cdots \underline{\alpha_{m-1}}$

When α_m is at the front enqueue # once more

$\triangleright \overline{\alpha_m} \# a \beta_1 \cdots \beta_n \$ \alpha_1 \cdots \alpha_{m-1} \#$

Dequeue and enqueue α_m

$\triangleright \overline{\#} a \beta_1 \cdots \beta_n \$ \alpha_1 \cdots \alpha_{m-1} \# \underline{\alpha_m}$

dequeue (#)

$\triangleright \overline{a} \beta_1 \cdots \beta_n \$ \alpha_1 \cdots \alpha_{m-1} \# \underline{\alpha_m}$

read front, dequeue, enqueue the read symbol till # is encountered at the front.

(....)

$\triangleright \overline{\#} \alpha_m a \beta_1 \cdots \beta_n \$ \alpha_1 \cdots \underline{\alpha_{m-1}}$

Dequeue #

$\triangleright \overline{\alpha_m} a \beta_1 \cdots \beta_n \$ \alpha_1 \cdots \underline{\alpha_{m-1}}$

(corresponds to $\triangleright \alpha_1 \cdots \alpha_{m-1} \underline{\alpha_m} a \beta_1 \cdots \beta_n \sqcup \cdots$ in standard TM.)

(2) Standard TM can do anything the queue-based deterministic TM does.

To access front, go to the left end of the input and read what the head pts to.

To access rear, go to the end of the input (special marker), go left and read the head's pointed cell.)

To enqueue, append at the end

To dequeue write $\sqcup$ to the first cell, shift content to the left by one.

e) $M = (K, \Sigma, \delta, s, H)$

where $K = \{s, q_s, q_a, q_b, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, y, n\}$,

$\Sigma = \{a, b, c, \triangleright\}$, // and may be $\sqcup$, not used here...

$H = \{y, n\}$, and

δ (state \ front)	a	b	c
s	(q_s, c)	(q_s, c)	(q_s, c)
q_s	$(q_a, \downarrow)$	$(q_b, \downarrow)$	$(y, \leftrightarrow)$
q_a	$(q_1, \downarrow)$	$(q_2, \downarrow)$	$(q_3, \downarrow)$
q_b	$(q_5, \downarrow)$	$(q_6, \downarrow)$	$(q_7, \downarrow)$
q_1	(q_a, a)	(q_a, a)	(q_a, a)
q_2	(q_a, b)	(q_a, b)	(q_a, b)
q_3	(q_4, c)	(q_4, c)	(q_4, c)
q_4	$(q_s, \downarrow)$	$(n, \leftrightarrow)$	$(n, \leftrightarrow)$
q_5	(q_b, a)	(q_b, a)	(q_b, a)
q_6	(q_b, b)	(q_b, b)	(q_b, b)
q_7	(q_8, c)	(q_8, c)	(q_8, c)
q_8	$(n, \leftrightarrow)$	$(q_s, \downarrow)$	$(n, \leftrightarrow)$

rear ← important only in one case otherwise taken as e

→ one case (rear: c front: c)

$\delta(q_s, c, a) = (n, \leftrightarrow)$

$\delta(q_s, c, b) = (n, \leftrightarrow)$

transitions associated with ▷ are omitted you may reject corresponding cases for simplicity as ▷ is not part of L's alphabet.

Read table as e.g.: $\delta(q_4, b, e) = (n, \leftrightarrow)$.

Rejection due to

i) mismatch btw corresponding characters

ii) different lengths of the first and the second substring on the left and right of 'c'.

e.g. Trace $(s, \triangleright \underline{b}abcba\underline{a}) \vdash_M^* (n, \triangleright \bar{a}c\underline{c})$ $\Leftrightarrow$ rejected

$(s, \triangleright \underline{b}abcba\underline{b}) \vdash_M^* (y, \triangleright \bar{c}\underline{c})$ $\Leftrightarrow$ accepted.

Q5) $L = \{ a^n b^{2^n} c^{3n} : n \in \mathbb{N} \}$

Idea: iteratively
- Decrement a's by 1
- Divide b's by 2
- Decrement c's by 3

} to reach some base case

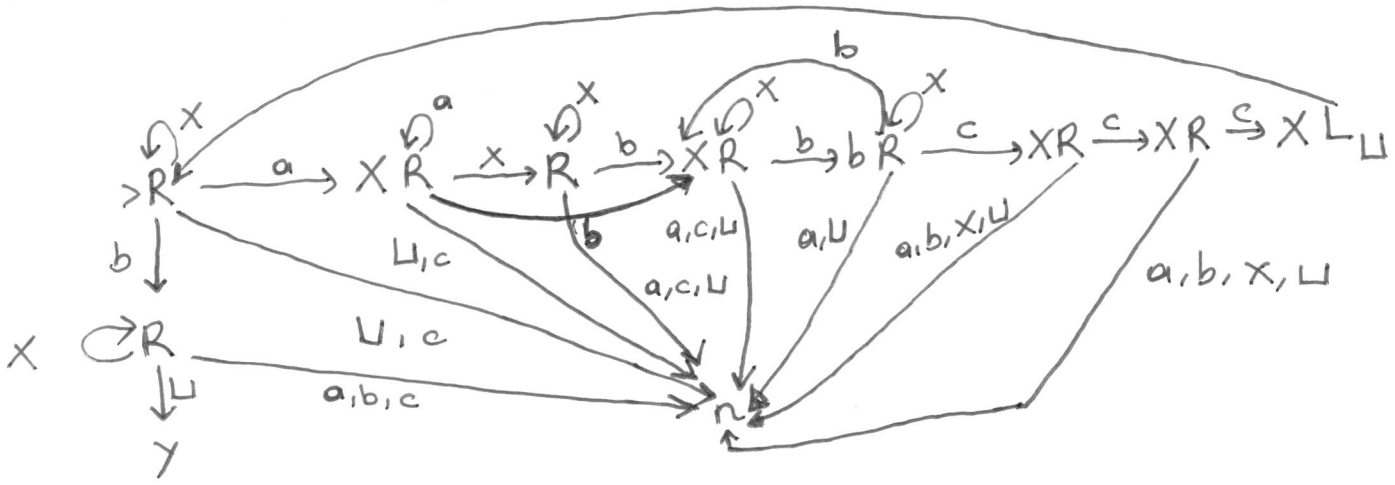
e.g. $a^3 b^{2^3} c^9$

	a a a	b b b b b b b b	c c c c c c c c c	$\rightarrow a^3 b^8 c^9$
Step 1.	X a a	X b X b X b X b	X X X c c c c c c	$\rightarrow a^2 b^4 c^6$
Step 2.	X X a	X X X b X X X b	X X X X X X c c c	$\rightarrow a^1 b^2 c^3$
Step 3.	X X X	X X X X X X X b	X X X X X X X X X	$\rightarrow$ b : base case ⇓ accept

Otherwise reject. → for every other string composition of a, b, c, X
→ if cross op^n fails.

Hence at each step
- → Cross an a (going over a section)
- → Cross every other b (going over b section)
- → Cross three b's (going over c section)

a) Some TM to decide (y or n) L:



(init config is $\triangleright \underline{\sqcup} w$.)

b) $G = (V, \Sigma, R, S)$

where

$V = \{S, M, P, \#, C, a, b, c\}$

$\Sigma = \{a, b, c\}$

$R = \{$ $S \to Mb\#$,
$M \to e \mid aCMP$,
$Pb \to bbP$,
$P\# \to \#$,
$\# \to e$,
$Ca \to aC$,
$cb \to bc$,
$bC \to bccc$,
$eC \to cccc$
$\}$

generate correct # of a's, b's, and c's

reshuffle them to reach correct order

Trying

$aCaCaCPPPb$ (exponentiate three times!)

organize those

$aaaccc\,b^8$

organize those

$aaab^7bCCC$

$bcccC$

$ccccC$

$cccc$

$a^3b^8c^9$.

Q6)

- L_1 is regular as regular grammar $\leftrightarrow$ FA.
- L_2 is deterministic context-free as there is a DPDA to parse its strings.
- L_3 is context-free.
- $\overline{L_4}$ does not have to be recursive / recursively-enumerable. Indeed, $\overline{L_4}$ can be any language whose intersection with the recursive language $\overline{L(a^*b^*)}$ (which is regular) yields a recursive language say, the empty set $\emptyset$, a regular language. Hence, in the most general sense $\overline{L_4}$ may not be recursively-enumerable. We know that there are ~~non~~ recursively-enumerable languages whose complement is also non-recursively-enumerable. In the broadest sense, L_4 can be an undecidable language.
- Although the mentioned grammar's language may not be L_5 as we do not know what other strings it generates, the statement suggests that all strings in L_5 can be enumerated by a finite representation. Consequently, L_5 is recursively-enumerable.

a. M_1, M_2, M_3, M_5 exist but M_4 may not exist.

b. L_1, L_2, L_3 are decidable languages. So, algorithms always exist to give y/n answers for arbitrary strings in their grammars' alphabets.

L_4 may be undecidable. In this case, no algorithm exists for its membership problem.

L_5 may be recursively-enumerable in the general case. If so, M_5 may fail to reject particular strings, thus no algorithm may exist for the corresponding membership problem.

c. $L = (\overline{L_2} L_1 \cap L_5)^* \cup (L_3^* \overline{L_4})$

recursive, recursive, rec-enumerable, recursive, undecidable?

recursively-enumerable

no abstract machine to recognize $\overline{L_4}$ (probably)
↳ no machine for L.

"Assume that $\overline{L_4}$ is rec-enumerable (or recursive) and try to construct a TM to recognize L which will be recursively-enumerable" Study for the final
(If you did that I will give most points, as the question could be misleading in its statement)

Study on how to prove that

Recursively-enumerable languages are closed under $\cup, \cap, ^*, \cdot$ and

Recursive languages are closed under $\cup, \cap, ^*, \cdot, -$

by constructing TM's.

d. L itself may be not recursively-enumerable. And $\overline{L}$ can also be so. We cannot always come up with a TM to semidecide $\overline{L}$.

Even when L is recursively-enumerable, $\overline{L}$ may not be recursively-enumerable as this class of languages are not closed under complementation. Thus, the answer is no.