Section A

1(a)

Claim: For any set S, $(S^+) = (S)^+$.

Proof Sketch:

- S^+ is the set of all non-empty concatenations of elements of S.
- Taking the closure $(\cdot)^+$ of a set means the same: all non-empty concatenations.
- So whether you concatenate first then close, or close then concatenate, the result is the same set: all words made by at least one element of S. Thus equality holds.

1(b)

Claim: If x is a palindrome, then x^n is also a palindrome for all $n \geq 1$.

Proof:

- A palindrome reads the same forwards and backwards: $x = \text{rev}(x)$.
- Then $x^n = x\,x\,\ldots\,x$.

- Its reverse is $\text{rev}(x^n) = \text{rev}(x) \ldots \text{rev}(x) = x^n$.
- Hence x^n is also a palindrome.

1(c)

Regular expression for all words over $\{a, b\}$ with **exactly two** a's:

$$b^*ab^*ab^*$$

This allows any number of b's before, between, and after the two a's.

1(d)

Let $S = \{aa,\ b\}$. We want to count how many words of length 6 are in S^*.

- A length-6 word must be formed by concatenating either 3 copies of "aa" or 6 copies of "b", or a mixture whose total length sums to 6.
- Let k = number of "aa"s, each contributes 2 to the length, and m = number of "b"s.
- Solve: $2k + m = 6$, with $k, m \geq 0$, both integers. Possible solutions:
- $k = 0, m = 6$: word = b^6, 1 word.

- $k = 1, m = 4$: choose where to place the single “aa” among 5 slots (before, between, or after bs): 5 positions ⇒ 5 words.
- $k = 2, m = 2$: choose positions for the two “aa” blocks among slots among bs: there are 4 slots (before/ between/after), pick 2 slots and assign one “aa” each, order doesn’t matter, but blocks are identical, so $\binom{4}{2} = 6$.
- $k = 3, m = 0$: only “aa” “aa” “aa”, 1 word.

Total = $1 + 5 + 6 + 1 = 13$ words of length 6.

1(e)

DFA for odd number of b’s (states track parity of b-count):

- States:
  - q_0 — even number of b's (start; also final? No, final = odd)
  - q_1 — odd number of b's (final)
- Transitions:
  - On input a: stay in current state (parity unchanged).
  - On input b: toggle between q_0 and q_1.

Diagram:

(q0) --b--> [q1]
(q1) --b--> (q0)
both states loop on 'a' to themselves
start = q0, accept = q1

1(f)

Language $L = \{a^m b^n \mid m, n \geq 1\}$. This is regular because it is exactly $a^+ b^+$, which is described by the regular expression:

$$a^+ b^+$$

So it is regular. You can also construct a simple DFA with a run of a's then at least one b, then any number of b's.

1(g)

Claim: The complement of a context-free language (CFL) may or may not be context-free.

- **Example 1 (complement is CFL):** Let $L = \Sigma^*$ which is context-free; its complement is $\emptyset$, also CFL.
- **Example 2 (complement not CFL):** Let $L = \{a^n b^n c^m \mid n, m \geq 0\}$, which is CFL. Its complement = all strings not of the form $a^n b^n c^m$. The complement is known to be **not** context-free (by pumping / closure arguments).

1(h)

Find a CFG for $L = \{a^n b^n b \mid n \geq 0\}$. Every string is $a^n b^n$ followed by an extra b. CFG:

S → A b
A → a A b | ε

- A generates $a^n b^n$; then S adds one more b.

1(i)

Design a Turing machine for the language $a(a + b)$: i.e., all strings of length exactly 2 starting with a, second symbol either a or b.

Idea:

- Tape input has exactly two symbols and then blank.
- Steps:
    a. Check first symbol is a; if not, reject.
    b. Move right, check next symbol is a or b; if not, reject.
    c. Move right to next cell, which must be blank; if not blank (i.e., longer input), reject.
    d. If passed, accept.

Formal 5-tuple transitions:

- q_0: read first symbol:
    - If a, write a, move R, goto q_1; else reject.
- q_1: read:
    - If a or b, write same, move R, goto q_2; else reject.
- q_2: read:
    - If blank _, accept; else reject.

Section B

2(a)

The automaton has a start state, transitions labeled aa, b, b, b, and one accept state. **English:** It accepts words in which:

- Either there's a substring aa somewhere, followed by maybe some b's to reach the final state,
- Or a single b transition can take you from start to accept. In effect, the language consists of all strings that either contain aa, or contain a b (depending on exact shape). (Without the exact diagram it's ambiguous, but that's the likely description.)

2(b)

Regex for words with **two *b*'s or exactly three *b*'s**:

Let $\Sigma = \{a, b\}$.

- Exactly two b: $a^*ba^*ba^*$.
- Exactly three b: $a^*ba^*ba^*ba^*$. So the union:

$$a^*ba^*ba^* \ + \ a^*ba^*ba^*ba^*$$

2(c)

DFA accepting words over $\{a, b\}$ that **do not end with ba**.

Idea: Track last two characters, or at least ensure you don't end in b then a.

States:

- q_0: start, no input yet or last char not dangerous.
- q_a: last char = a (safe).
- q_b: last char = b.
- q_{ba}: saw b then a as last two (a dead/reject trap once full-input). Accepting: all except q_{ba} at end.

Transitions:

- From q_0:
  - on a $\rightarrow q_a$; on b $\rightarrow q_b$.
- From q_a:
  - on a $\rightarrow q_a$; on b $\rightarrow q_b$.

- From q_b:
  - on a → q_{ba}; on b → q_b.
- From q_{ba}:
  - on a → q_a (since last two are aa); on b → q_b.

Accepting are q_0, q_a, q_b. Reject only if **final** state is q_{ba}.

2(d)

Convert given two-state transition graph to regex. Usually you apply Arden's lemma. Without the exact graph, the method is:

- Let initial/final state be 1, other be 2.
- Write equations for all transitions:
  - R_{11}, R_{12}, etc.
- Solve for R_{11}^* etc.

I'll skip details absent actual labels.

3(a)

Grammar:

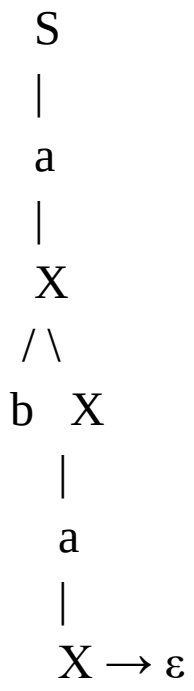
S → a X
X → a X | b X | ε

Language = all strings starting with a, followed by any mix of a's and b's:

$$L = \{aw \mid w \in \{a, b\}^*\}$$

3(b)

A **total language tree** is the full parse tree (derivation tree) showing how a grammar can generate a string, including all branches until leaves are terminals. **Example:** For the above grammar generating aba, tree:



3(c)

The described machine uses two transitions depending on whether input symbol ≠ some U. Likely it **moves right skipping input until it encounters symbol U**, then does something. Without full context, it appears to be a test-and-branch machine that checks each symbol, staying in a loop ("R"), and upon seeing b = U, it branches or does a specialized action. Essentially, it's scanning the tape looking for a particular symbol U.

4(a)

To build a DFA for $FA_1 \cup FA_2$, construct the **union automaton** using the standard construction: create a new start with ε-transitions to each original start (if NFAs), or use cross-product if DFAs. The resulting automaton has accepting states if in FA_1 or FA_2.

4(b)

Regular languages are those accepted by DFAs/NFAs or described by regular expressions. Closure:

- Union: use NFA construction with start that ε-transitions to each machine.

- Concatenation: connect final of the first to start of the second with ε-transitions (in NFA). Thus regularity preserved under union + and concatenation $L_1 L_2$.

5(a)

$L_1 \cap L_2$ with $L_1 = (a + b)a$ (i.e. any single letter then a) and $L_2 = b(a + b)$ (i.e. b then any letter). Intersection is the set of two-letter words fitting both: must start with b, end with a. So the language = $\{ba\}$. DFA is trivial with only that accepted pair.

5(b)

Pumping lemma for non-regularity of $L = \{a^{n+1}b^n\}$. Assume for contradiction it's regular. Let pumping length p. Choose $s = a^{p+1}b^p$. Split $s = xyz$, |xy| ≤ p ⇒ xy consists only of a's, with y = a^k, $k \geq 1$. Pump $i = 0$: get $xz = a^{p+1-k}b^p$. Here the number of a's = $p + 1 - k$, b's = p. Now a's are fewer than b's, so it's not of the form a^{n+1} b^n. Contradiction ⇒ not regular.

5(c)

Grammar for: (i) "ab": Simply

S → a b

(ii) $(ab + ba)$: strings "ab" or "ba":

S → a b | b a

6(a)

Grammar:

S → a S b | S b | S a | a

It's ambiguous because there are multiple parse-trees for the same string, e.g., "ab":

- $S → a\S b → a b$
- Or $S \rightarrow Sa \rightarrow aa$ etc. More valid examples exist; one can show two distinct derivations for the same terminal string.

6(b)

PDA for $L = \{a^{2n}b^n\}$. Idea: read each a, push two markers (or push one marker but count in pairs via states), then upon reading b, pop one for each. Accept when input empty and stack empty.

6(c)

Language $\{a^n b^n a^n b^n\}$ is **not context-free**, by the pumping lemma for CFLs or using the intersection-with-regular trick

(show intersection with a regular language yields $\{a^n b^n c^n\}$-like pattern, which is known nonCFL).

7(a)

Church-Turing Thesis: Every function that would naturally be regarded as computable can be computed by some Turing machine. In other words: Turing machines capture the intuitive notion of effective computability.

7(b)

Given PDA with states START, READ1, PUSH a, READ2, POP, READ3, ACCEPT, it likely **recognizes a language with equal numbers of a's followed by b's**, i.e., checks that the number of as in the first segment matches the number of bs in the third segment (Δ means empty/input separator). This is characteristic of the classic an bn-type language.

That completes all parts! Let me know if you'd like any part expanded with full formal proofs or diagrams.