1. Convert Context Free Grammar to Chomsky Normal Form (CNF):

Given Grammar:

VN = {S, A, B}

VT = {a, b}

P:

S → a B

S → B A

A → a

A → b B

A → A a b B

B → ε

B → b S

B → a A B A

Step 1: Eliminate ε-productions:

Replace B → ε with B → SS | SB | BS | SA | AS.

Step 2: Eliminate unit productions:

There are no unit productions in this grammar.

Step 3: Convert productions to have at most two symbols on the right side:

Rewrite long productions by introducing new non-terminals:

S → BA | AB

A → a

A → bB

A → AAB

B → bS

B → aAB | aBA

The CNF version of the grammar is now in the specified form.

2. Convert Grammar to Greibach Normal Form (GNF):

Given Grammar:

VN = {S, A, B, C}

VT = {a, b}

P:

S → bB

A → BA

B → AC

A → B

A → b

C → a

Step 1: Each production's right-hand side should start with a terminal symbol.

Rewrite the productions accordingly:

S → bB

A → bA | BA

B → aC

A → B

A → b

C → a

The grammar is now in Greibach Normal Form.

3. Construct Pushdown Automata for the language L:

The language L = {a^(2m) b^(n) a^(n) | m, n ∈ N} represents strings with twice as many 'a's followed by 'b's, and then followed by 'a's equal to the number of 'b's.

Pushdown Automaton (PDA):

States: {q0, q1, q2, q3}

Alphabet: {a, b}

Stack Alphabet: {Z0, A}

Transition Rules:

(q0, a, Z0) -> (q0, AZ0)

(q0, a, A) -> (q0, AA)

(q0, ε, A) -> (q1, ε)

(q1, b, A) -> (q2, ε)

(q2, ε, Z0) -> (q3, ε)

Analysis of the Word:

For the word "aabbaa", the PDA would start at q0, pushing 'A' onto the stack for each 'a', transitioning to q1 after reading all 'a's, then popping 'A' for each 'b' in q2, and finally transitioning to q3 after reading all 'b's, where the stack becomes empty.

4. Matrix of Simple Precedence & Derivation Tree:

Given Grammar:

VN = {S, A, B, C, D}

VT = {a, b, c, d, e, f}

P:

S → A

A → aD

D → b

D → bD

D → Ac

Matrix of Simple Precedence:

	b	c	d	e	f	
a		<	<	<	<	<
b	>	=	<	<	<	<
c						
d	>					
e						
f	>					

Analysis of "ababbc" and Derivation Tree:

Step 1: S → A

Step 2: A → aD

Step 3: D → b

Step 4: D → bD

Step 5: D → Ac (Apply this rule to generate the remaining 'c')

Derivation Tree:

```
    S
    |
    A
  / | \
 a  D  c
    |
    b
    |
    b
```