1. Exercise 1

1.1
The only gates that need to be decomposed are the OR3 and NAND3 gates.

Their decomposition is as follows:

Where:

1) Decomposition of OR3 to OR2.  \( OR3 = (a + b + c) = ((a + b) + c) = OR2(OR2(a, b), c) \)

2) Decomp. of OR2 to NOT and NOR2.  \( OR2 = (a + b) = (a + b) = NOT(NOR2(a, b)) \)

3) Replacing all the OR2 gates with the Decomp. from (2).

4) Decomp. of NAND3 to AND3 and NOT

5) Decomp. of AND3 to 2 AND2’s.  \( AND3 = abc = (ab)c = AND(AND(a, b), c) \)

6) Decomp. of AND2 to 2 NOT’s and NOR2.

7) Replacing all the AND2 with the Decomp. from (6).

The overall decomposed gate is presented on the next page.
We notice the yellow pair of inverters and remove them, as it doesn't change the logic.
The final circuit is:

1.2 Identification of the library patterns is as follows:
In variation A there were found 5 NOT gates, 1 NOR3 pattern (in green) and 1 NOR4 pattern (in red)
In variation B there were found 6 NOT gates, 3 NOR2 gates and 1 AOI21 pattern (in yellow)
In variation C there were found 6 NOT gates, 1 NOR2 gate and 2 NOR3 patterns.

1.3
The only effective place to add a pair of inverters is where we first removed them in section 1.1, since this
is the only place where a NOR2 is connected to another NOR2 and not NOT. Inserting the pair in other
places won't have any effect as there is an inverter in each node.

Now we can see that there no new possible patterns from the library can be identified. In Fact, the AOI21 pattern
(seen in yellow in variation B) is now impossible to be implemented.
To summarize, adding a pair of inverters to the circuit doesn't help.

1.4
In the trivial case, using only NOT's and NOR2's, we have in total, 8 NOT gates (each costs 2) and 5 NOR2
gates (each costs 3), so the trivial cost will be:

\[
\text{trivialCost} = 8 \cdot 2 + 5 \cdot 3 = 31
\]
1.5
We will perform the covering on circuit 1.1, Q2, (as 1.3, Q3, is not helpful, anyway covering on 1.1 or 1.3 should give identical results as unhelpful NOT pairs must be removed).

As we were not asked to find the optimal solution we will use the variations we found on 1.2.

Variation A:

As we can see, the cost of Var. A is 19 which is better than 31 that we found in 1.4.

Variation B:

As can be seen, the cost of Var. B is 25 which is better than 31 that we found in 1.4, though worse than Var. A.

1.6
All the gates in the circuit are either NOR2 or NOT, so there is no decomp. required.

There are 2 NOT's and 3 NOR2's, so the cost of the circuit is:

\[
\text{trivialCost} = 2 \cdot 1 + 3 \cdot 2 = 8
\]
1.7
Using the pattern of AOI21 to cover the circuit doesn't improve the cost. The circuit, as is, doesn't have AOI21 pattern, and in order to add "by force" we can try to add a pair of inverters to match the pattern. The AOI21 pattern includes two NOR2, one connected to the other, so the suitable place to put the pair of inverters will be:

(In yellow, the add inverters, in green the AOI21 pattern)
In this variation the total cost is 8 which is the same as the cost in 1.6, so this addition didn't help.

Another suitable place will be:

In this variation we also get a total cost of 8 thus, we conclude that our cost cannot improve with the use of AOI21 cell (we can try and add NOR2 cells but that will complicate the circuit and will give higher costs if not 8).

1.8
For a given gate composition, dynamic programming for tree covering assures that the solution reaches the best optimal value, and not a local extreme of the problem. Thus, there is no meaning for simulated annealing to this kind of problem, as it provides some random solution which may or may not be even close to the optimal solution. However, there is a possibility to implement the "idea" of simulated annealing on the gate composition of the circuit. Meaning, to add pair off inverters or other gates to the circuit such that the logic function doesn't change, but new patterns which weren't available in the previous comp. are now available. Thus, the trivial cost of the circuit may rise, but the optimal solution may be better. Of course the use of simulated annealing will be with dynamic programming, and not alone.