1 Horn-SAT

**Problem statement**: Given a set of Horn clauses, determine whether there exists an assignment to variables such that all clauses are satisfied.

**Proof**:
If the algorithm outputs a solution, by design of algorithm, the solution must satisfy all clauses \((x_1, x_2, ..., x_n)\).

If the algorithm outputs no solution, assume towards contradiction, there is a satisfying assignment \((x'_1, x'_2, ..., x'_n)\). Let \(i_1, i_2, ..., i_k\) be the ordering in which the algorithm sets the variables to be true.

1. If \((x'_{i_1}, x'_{i_2}, ..., x'_{i_k})\) are all true, let \(C\) be the type 3 clause that assignment \((x_1, x_2, ..., x_n)\) violates, the variables in \(C\) must in \((x'_{i_1}, x'_{i_2}, ..., x'_{i_k})\). Since \(X'_{i_j}\) is also true for \(j = 1, 2, ..., k\), \(C\) must be violated by \(X'_{i_j}\). Thus, there is a contradiction.

2. Let \(i_j\) be the first variable where \(X_{i_j}\) is true and \(X'_{i_j}\) is false. When \(X_{i_j}\) were set to true. There are two possible cases.
   (a) \(X_{i_j}\) is set to true by a type 2 clause.
   (b) \(X_{i_j}\) is set to true by a type 1 clause.

In both sub-cases, this particular clause will be violated by \((x'_j)\). Thus, there is a contradiction.

2 Huffman Tree

**Problem statement**: Given a long string with \(n\) different characters in alphabet, find a way to encode these characters into binary codes that minimizes the length.

**Algorithm**

1. REPEAT
2. Select two characters with smallest frequencies
3. Merge them into a new character, whose frequency is the sum.
4. UNTIL (there is only one character)

**Running Time:**

1. Naive implementation: $O(n^2)$.
   - $n - 1$, every iterations reduces number of characters by 1
   - $O(n)$ for each iteration.

2. Priority Queue/Heap Implementation: $O(n\log n)$

**Proof Of Correctness:**

*Induction Hypothesis:* Huffman Tree algorithm finds an optimal encoding for all alphabets of size at most $n$.

*Base Case:* When $n = 1$, there is only one solution with cost 0.

*Induction Step:*

Assume induction hypothesis is true for $n$, consider an alphabet of size $n + 1$, assume towards contradiction that Hoffman Tree algorithm does not find the optimal solution, let $T_{alg}$ be the tree found by the algorithm and $T_{opt}$ be the tree found by OPT, and $i, j$ be the first two characters that the algorithm merged.

If $i, j$ are not children of the same node in $T_{opt}$:

Let $i', j'$ be the two nodes at the highest depth in $T_{opt}$ that share the same parent. Let $T'_{opt}$ be a solution where $i$ and $j$ are swapped with $i'$ and $j'$ in $T_{opt}$.

Let $d_i$ be the depth of $i$ in $T_{opt}$, and similarly for $d_j, d_{i'}$ and $d_{j'}$. We have thus:

$$
\text{cost}(T'_{opt}) = \text{cost}(T_{opt}) - (W_i * d_i + W_j * d_j + W_{i'}d_{i'} + W_{j'} * d_{j'}) + (W_i * d_{i'} + W_j * d_{j'} + W_{i'd_i} + W_{j'd_j})
$$

$$
\leq \text{cost}(T_{opt}) - (W_{i'} - W_i)(d_i - d_{i'}) - (W_{j'} - W_j)(d_j - d_{j'})
$$

Therefore, $T'_{opt}$ is also an optimal solution.

Now that we know there is always an optimal solution that merges $i$ and $j$, the problem reduces to an alphabet of size $n$. By induction hypothesis, Hoffman tree algorithm is optimal for this instance. Therefore, $\text{cost}(T_{alg}) < \text{cost}(T_{opt})$. Thus, it contradicts with the assumption that $T_{alg}$ is optimal.