- **reduction**
  - Problem A can be reduced to problem B, if one can design an algorithm for problem A using an algorithm for problem B as a subroutine.
  - If A can be reduced to B, A is "easier" than B, \( A \leq B \).

- **Example**
  - **LIS**: input \( \{4, 5, 2, 3, 6, 9, 7\} \)
    - Goal: find the longest subsequence, increasing \( \{2, 3, 6, 7\} \)
  - **LCS**: input \( \{2, 4, 1, 5, 4\} \), \( \{2, 5, 4, 3, 1\} \)
    - Goal: find longest subseq that is a subseq of both inputs \( \{2, 5, 4\} \)

```java
LIS (X)
\{ 
  Y = sort(X) ; 
  return LCS(X, Y) ; 
}\n
X = \{4, 5, 2, 3, 6, 9, 7\} 
Y = \{2, 3, 4, 5, 6, 7, 9\} 
\{2, 3, 6, 7\}
```

Proof: 1) Every common subseq. of X and Y is an increasing subseq. of X.

2) Every increasing subseq. of X is also a common subseq. of X and Y.

\( \text{LIS} \leq \text{LCS} \)

- **complexity**
  - \( P \) (Polynomial time): class of decision problems that
can be solved in polynomial time.

- **NP** (nondeterministic polynomial time) set of decision problems that can be verified in polynomial time.

Verify (input, answer, explanation)

```plaintext
if answer == YES
    if explanation supports answer
        accept
    else
        reject

run in polynomial time
```

if true answer is yes, then there exists an explanation that makes verify(1) accept.

if true answer is no, then no matter what explanation verify(1) will reject.

- **NP-complete Problem**

a problem B is **NP-complete** if

1. **B** ∈ **NP**
2. for every **A** ∈ **NP**, there is a polynomial time reduction from A to B. (A ≤ B)