Basics of Algorithm Analysis. (Asymptotic Notation)

What are *efficient* algorithms?

- First attempt: When implemented, the algorithm runs fast on real inputs.

Stable Matching: \( n = \# \) of men = \( \# \) of women.

- What is the running time* as a function of \( N \)?

  Lucky: \( n \) proposals.

  Unlucky: \( \approx n^2 \) proposals.

- Worst-case running time.

Example: Algorithm A

\[ \text{\# of operations:} \quad N + 1000 \]

\[ N \geq 5 \quad \text{A is slower.} \quad N = 10^6 \quad \text{then B is slower.} \]

Asymptotic Order of Growth. (Big-O notation).

Definition 1. Let \( T(n) \) be a function. Given a function \( f(n) \), we say \( T(n) \in O(f(n)) \) iff

\[ \exists \text{ constants } C \text{ and } n_0 \text{ such that } \]

\[ \forall n \geq n_0, \quad T(n) \leq C \cdot f(n) \]

Example 1. \( 5n^2 + 3n + 1 = O(n^2) \).

Proof: \( n_0 = 1 \). \( c = 5 + 3 + 1 = 9 \).

\[ 5n^2 + 3n + 1 \leq 5n^2 + 3n^2 + n^2 \]

\[ = 9n^2 \leq cn^2. \]
Example 2: $5n^2 + 3n - 1000 = O(n^2)$?

True. $n_0 = 1, c = 8$.

Example 3: $n \log_2 n = O(n)$? False!

we can show no

$(n_0, c)$ would work.

Fix any $n_0$ and $c$.

Let $n = \max(n_0, 2^c) + 1 > n_0$

$n > 2^c \Rightarrow n \log_2 n > n \cdot c \Rightarrow$ contradiction!

Reasons for using big-\(O\):

For $i = 1$ to $n-1$

for $j = i+1$ to $n$

- do something

\[ T(n) = (n-1)+(n-2)+\ldots+1 = \frac{n(n-1)}{2} = O(n^2). \]