(3) Time Complexity I - Big-O Notation

Def'n 3.1 (Running Time, Input Size)
We define the running time to be the # of elementary operations (i.e. arithmetic ops, comparisons, assignments) executed by an algorithm.
This will commonly be represented as a function of input size, which is the # of bits needed to represent the input

Def'n 3.2 (Big-O Notation)
Given $f: \mathbb{R}_+ \to \mathbb{R}_+$, $g: \mathbb{R}_+ \to \mathbb{R}_+$, we say that:
$f(n) \in O(g(n))$
if there exists constants $c \geq 0$, $n_0 \geq 0$ s.t.
$f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$

Ex:
1) $n \in O(2n)$ since $n \leq 1 \cdot 2n \ \forall n \geq 0$
$2n+10 \in O(n)$ since $2n+10 \leq 3 \cdot n \ \forall n \geq 10$

2) $3n \in O(n^2)$ since $3n \leq 3n^2 \ \forall n \geq 0$
But,
$n^2 \notin O(3n)$
In general:
If $0 \leq \alpha_1 < \alpha_2$ and $\beta_1, \beta_2 \geq 0$:
$\beta_1 n^{\alpha_1} + \beta_2 \in O(n^{\alpha_2})$
But,
$n^{\alpha_2} \notin O(n^{\alpha_1})$

Continued

Ex: Continued)

3) $n\log_2(n) \in O(n^2)$, but not $O(n)$

4) $\log_2 n = \frac{\log_c n}{\log_c 2} = \frac{\log_d n}{\log_d 2}$

So, in general, for any $c, d \geq 1$:

$$\log_c n \in O(\log_d n)$$

5) $2^n \in O(3^n)$, but $3^n \notin O(2^n)$

6) $f(n) \in O(c) \equiv f(n) \leq c \quad \forall n \geq n_0$.

i.e. This is shorthand for saying $f(n)$ is bounded by a constant)

Note:

Using the notation in (6), we write for: "$f(n)$ is bounded by some polynomial of n":

$$f(n) \in O(n^{O(1)})$$

Def'n 3.3 (Efficient)

An algorithm with running time $f(n)$ is efficient, if $f(n)$ is bounded by some fixed polynomial of n.

→ Analysis of our algorithms.

For SPT, WSPT, EDD, these algorithms involve sorting n numbers (jobs). So these require $O(n \log n)$ time

For LCL, we perform the analysis below:
There are n jobs, so:
- We perform n iterations (exactly)
- Each iteration $i = 1, \ldots, n$ requires finding the minimum of $(n-i+1)$ $f_j(\cdot)$'s at a given time t.

We can consider $f_j(\cdot)$ to be an elementary operation, and so LCL will have $O(n^2)$ running time:

Note:
Trying out all possible solutions is NOT an efficient algorithm, as this requires $n!$ time, but $n! \in O((\frac{n}{e})^n)$, and so it is NOT bounded by a poly in n.