CS 135 — L09: Efficiency Patterns

CS 135 Notes • 2025-10-21

Website: student.cs.uwaterloo.ca/~cs135

Linear (one recursive call)

```
(define (appnd xs ys)
  (cond [(empty? xs) ys]
        [else (cons (first xs) (appnd (rest xs) ys))]))
```

Quadratic (linear helper inside linear recursion)

Insertion sort:

```
(define (insert n xs)
  (cond [(empty? xs) (list n)]
        [(< n (first xs)) (cons n xs)]
        [else (cons (first xs) (insert n (rest xs)))]))

(define (sort xs)
  (cond [(empty? xs) empty]
        [else (insert (first xs) (sort (rest xs)))]))
```

`insert` is linear; called once per element ⟶ $\mathbf{O(n^2)}$.

Avoid Exponential Branching

```
(define (larger a b) (if (> a b) a b))
(define (largest lst)
  (cond [(empty? (rest lst)) (first lst)]
        [else (larger (first lst) (largest (rest lst)))]))
```

⚠️ **Warning** — Exponential growth often appears when you make **two recursive call** on **overlappinh** sublists.

Practice

- Identify linear/quadratic parts of your own code; refactor quadratic patterns using accumulators.