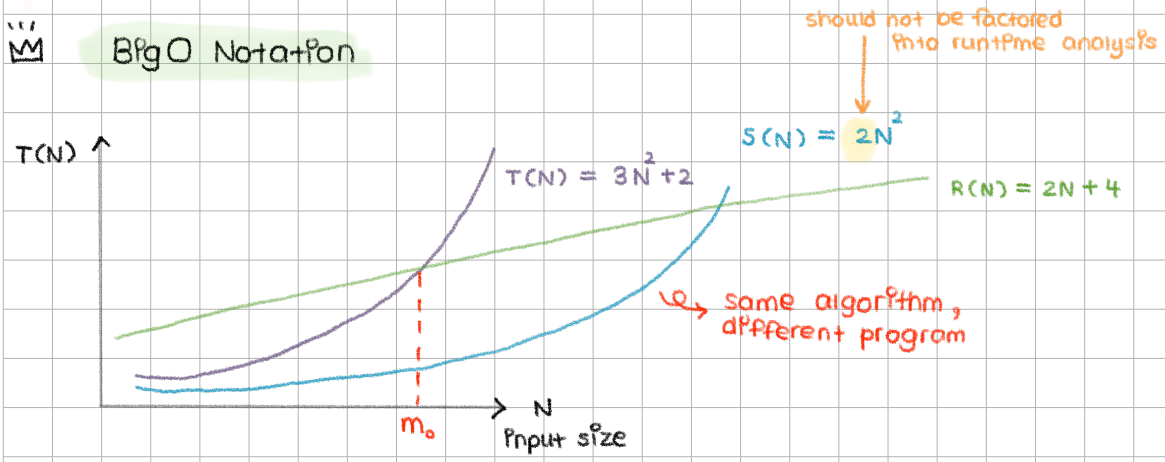
Big O Notation



Ideas:

→ Compare functions regardless of constants

$f(N) = O(g(N))$ iff $f(N) \geq c \cdot g(N)$ for all $N \geq m_0$

for some constant c at most m_0

$R(N) = O(T(N))$ → "upperbound" (It's not going to be worse than this)

$f(N) = \Omega(g(N))$ iff $f(N) \geq c \cdot g(N)$ for all $N \geq m_0$

* comparison-based sorting is $\Omega(N \log N)$

Selection Sort

p ↓

find min to the right of p

swap element at p with that min

```
for p in range(0 ... N):
    for i in range (p, N):
```

$N + (N-1) + (N-2) + \dots + 1 = \frac{N(N-1)}{2} = \frac{N^2 - N}{2} = O(N^2)$

Insertion Sort

p ↓

$1 + 2 + 3 + \dots N \Rightarrow O(N^2)$

* just like selection sort but reversed

```
for p in range (1, N):
    i = p
    while li[i] < li[i-1]
        swap (i, i-1)
        i -= 1
```

worst case: $O(N^2)$

best case: $O(N)$

Sorting Stability

1_A 3 1_B 2 3 2

- Stable = Relative order of the list should be the same
- Why does it matter?
  → Email - Sort by the same sender (emails there should be the same)
- Selection Sort is not stable (same values can be out of order)
- Insertion Sort is stable

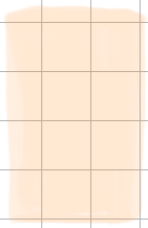
Stacks and Queues → Runtime for all stack and queue operations are constant

list: [ ]

append, insert (e, x), remove (e)

Python lists are arrays!

Stacks "LIFO" storage



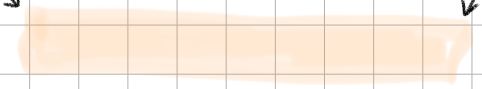
Two main operations: push() and pop()

Applications: Palindrome detection
→ push first half
→ pop and compare with the second half
→ If odd length, skip middle

⚠ Lists in Python can be used as a stack
→ push() = append()
→ pop() is the same

Queues "FIFO" storage → In Python, just push(x) and pop(0)

front ↘ end ↙

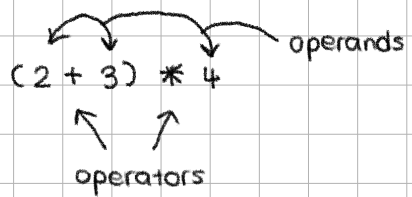


Two main operations:
enqueue(x): add x at the end
dequeue(): return and remove the element at the front

Applications: Scheduling and shared resources
Process scheduling in OS (If you only have 1 CPU)
↳ Round-robin scheduler

Reverse Polish Notation

Standard:



operand, operator, operand

RPN:

2 3 + 4 *

operand, operand, operator

Evaluating RPL expressions

```
S = 2 3 + 4 *

    If S[i] is a number:
        stack.push(S[i])
    else:  # operator
        operand2 = stack.pop()
        operand1 = stack.pop()
        stack.push(operator(operand1, operand2))
    return stack.pop
```