Rebalancing

```
def rebalance(self):
    if self.slope() == 2:
        if self.left.slope() == -1:
            self.left.leftrotate()
        self.rightrotate()
    ⋮
```

self.slope()?

height(left) - height(right)

Compute self.height() recursively

Takes O(size)

Soln: Add a local quantity height in each node, init=1 (or 0)

Update locally with each modification

Running time

- How long it takes, as a function of input size
- What to measure?

$x = x + 5$ ⇝ load x from memory
write 5
replace back

Assume some convenient (reasonable) notion of basic step

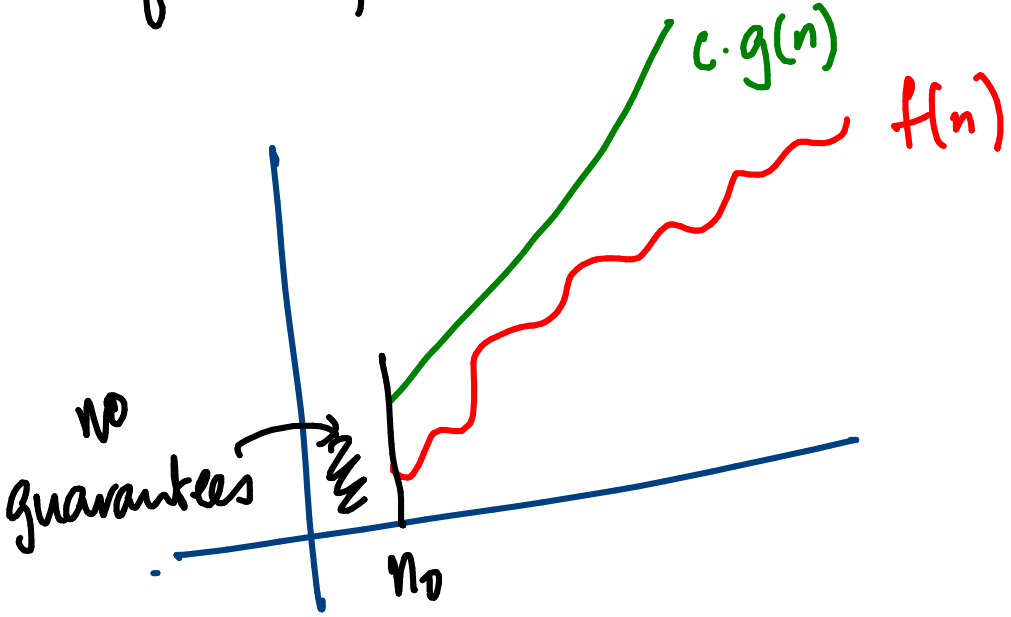
Arithmetic op
Comparison, Assignment

- Running time as a fn of input size, ignore constant factors
- Not all inputs of same size behave same
  - "Average" across all inputs of size n
    - Need a probability distribution over input space
  - Worst case
    - e.g. search in unordered list – not found
      scan entire list

Notation $O(\,)$

$f(n)$ is $O(g(n))$

if $\exists n_0, c$ s.t. $\forall n \geq n_0 \; f(n) \leq c \cdot g(n)$



$f(n) = 3n^2 + 6n + 22$ $g(n) = n^2$

$n \geq 1$ $f(1) = 31$ Take $c = 31$

$3 \cdot 2^2 + 6 \cdot 2 + 22$ $31 \cdot 2^2$

$3 \cdot 2^2 + 6 \cdot 2^2 + 22 \cdot 2^2$

$an^2 + bn + c$, $a, b, c > 0$

$\Rightarrow$ Choose $n_0 = 1$, constant $= a + b + c$

Negative coeff - drop them!

"Highest term" dominates

$$\log n < n < n^2 < n^3 \cdots 2^n$$

In practice

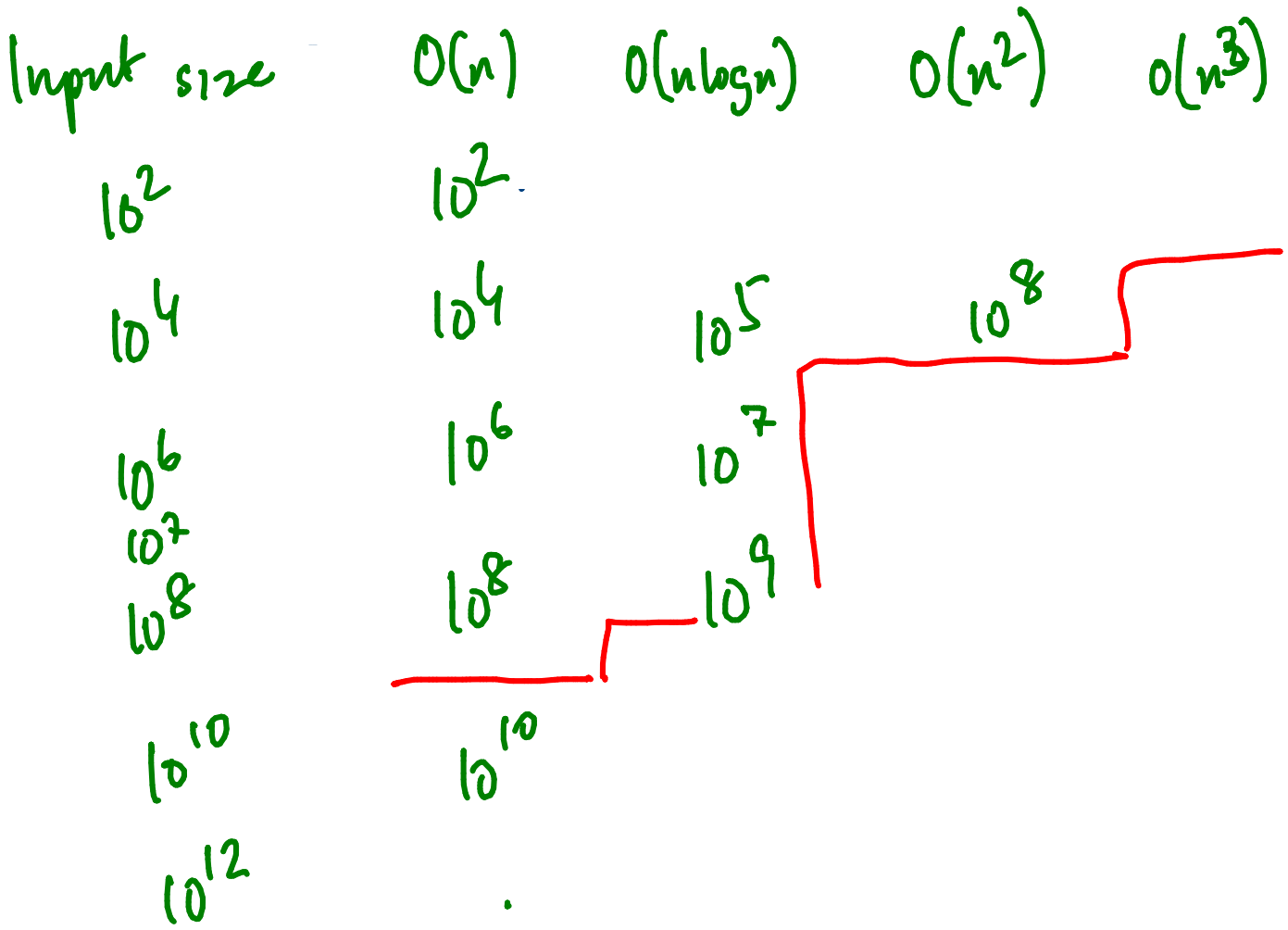
Concrete assumptions about basic step

Typical PL ops arithmetic, compare, assign

10^8 operations/second

With small constants

$2^{10} \sim 1000$



Sorting

Insertion Sort

Merge Sort

Quick Sort

```
isort [] = []
isort (x:xs) =
    insert x (isort xs)
```

What is worst case behaviour of isort()?

Recurrence

$$T(0) = 1$$

$$T(n) = T(n-1) + (n-1)$$

isort (x:xs) isort xs insert

$T(0) = 1$

$T(n) = T(n-1) + (n-1)$

$0\ 1\ 2\ \ldots\ n-1$

$n-1\ \ldots\ \ldots\ 0$

$n-1\ n-1\ \ldots\ n-1$

$= T(n-2) + (n-2) + (n-1)$

$= T(n-3) + (n-3) + (n-2) + (n-1)$

$\vdots$

$= T(0) + 0 + 1 + \ldots + (n-1)$

$= 1 + 0 + \ldots + (n-1)$

$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$

$\frac{n(n-1)}{2}$

Binary Search (Arrays)

Check midpoint

Search left or right half

$O(1)$ means constant time

$T(1) = 1$

$T(n) = 1 + T(n/2)$

1: check midpoint

$T(n/2)$: left or right half

$= 1 + 1 + T(n/4) = 1 + 1 + T\left(\frac{n}{2^2}\right)$

$= k + T\left(\frac{n}{2^k}\right)$ → becomes 1 when $k = \log n$

$\log n$ ←

Selection Sort

Select minimum $O(n)$

Move to sorted list $O(1)$

$T(0) = 0$

$T(n) = n + T(n-1)$

select rest

Similar to Insertion Sort $O(n^2)$

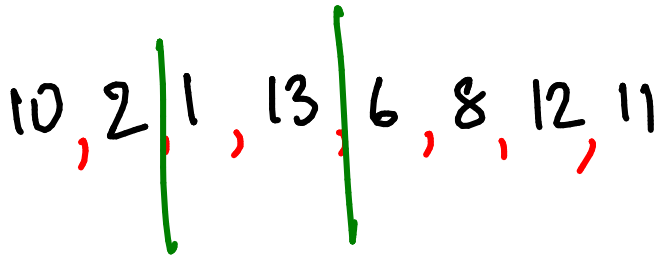
Merge Sort

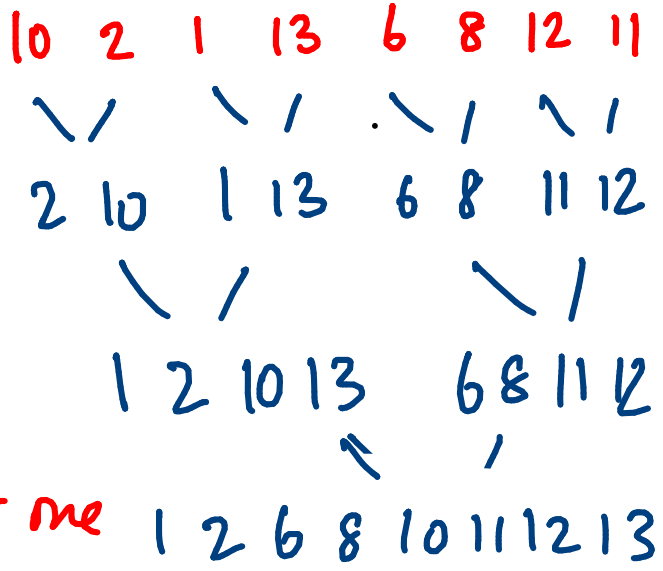
Split into two halves

Sort each half

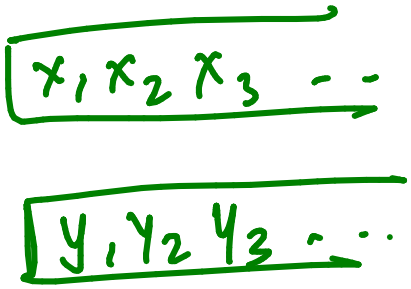
Merge

Recurrence ?





Merge



Compare x_1, y_1, move out smaller one

Repeat