Asymptotic notations are used to describe running times of algorithms.

| $\Theta$ | $O$ | $\Omega$ | $o$ | $\omega$ |
| --- | --- | --- | --- | --- |
| Theta | Big O | Omega | little o | little omega |
| $=$ | $\leq$ | $\geq$ | $<$ | $>$ |

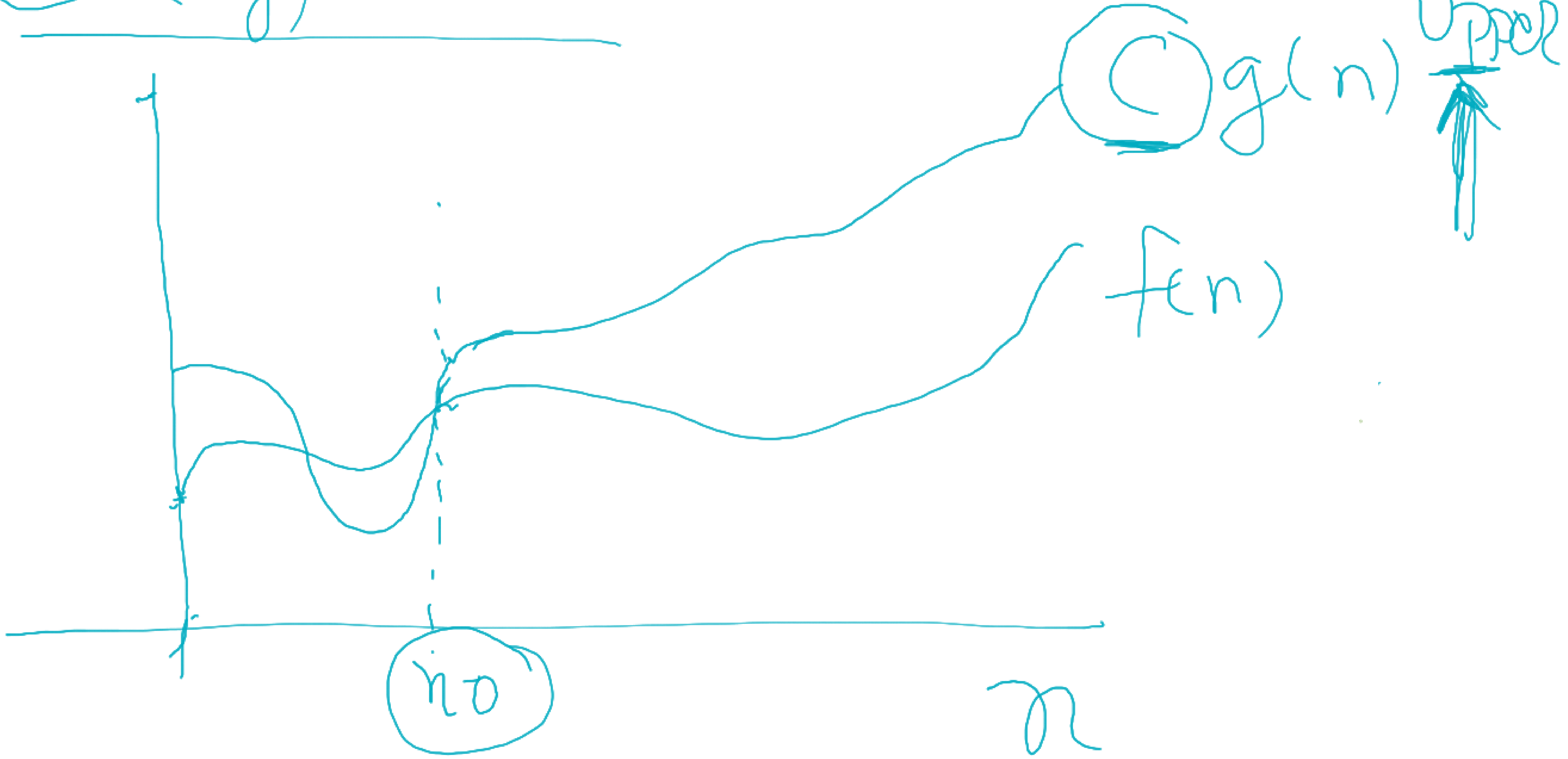
Analogy

To give idea

# Θ Theta Notation

$$f(n) = \Theta(g(n))$$

# O (Big) Notation $f(n) = O(g(n))$



# Ω (omega) Notation

Ex.

1) Logarithms grow slowly than polynomials.

2) Polynomials grow slowly than exponentials.

* changing the base of a logarithm from one constant to another constant changes the value by a constant factor, so we don't worry about logarithm bases in asymptotic not".

# Relational Properties

Transitivity
Reflexitivity
Symmetric
Transpose

| Trichotomy

statement:
f(n) is asymptotically smaller than g(n)
if $f(n) = \underset{\text{little}}{o}(g(n))$

f(n) is asymptotically larger than g(n)
if $f(n) = \omega(g(n))$

$\omega(g(n)) < fn < \underset{\text{little}}{o}(g(n))$

What is recurrence?

Recurrence is a function defined in terms of

- one or more base cases and
- itself, with smaller arguments

Recurrence for binary search

$T(n) = T(n/2) + \Theta(1)$ ← Base case

variable indicating size

# Substitution method

1. Guess for $sol^n$
2. prove using PMI

$T(n) = 2T(n/2) + n \quad n>1$ | PMI

$T(n) = \Theta(n \log n)$

① Base case $= 1, \ n=1$

$n=2 \Rightarrow n \log n \Rightarrow 2 \log_2 2 \Rightarrow 2 \cdot 1 \Rightarrow 2$

② Inductive step

hypothesis is that $T(k) = 2T(k/2) + k$

k to be $n/2$

$T(n/2) \Rightarrow \Theta\left(\frac{n}{2} \log\left(\frac{n}{2}\right)\right)$

③

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

$= 2\left(\frac{n}{2} \cdot \log\left(\frac{n}{2}\right)\right) + n$ ∵ From Hypothesis

$= n\left(\log n + \log \frac{1}{2}\right) + n$ ∵ $\log_a mn$

$\Rightarrow \log_a m + \log_a n$

$$\therefore -\log_b (1/a) = -\log_b a$$

$$= n(\log_2 n - \log_2 2) + n$$

$$= n(\log_2 n - 1) + n$$

$$= n\log_2 n - n + n$$

$$= n \log n$$

Comparison of insertion sort with merge sort

- On small inputs, insertion sort may be faster

But for large enoug inputs, merge sort will always be faster

Because, merge sort's running time $\Theta(n \lg n)$ grows more slowly than insertion sort's running time $\Theta(n^2)$.

Assignment: $n^{th}$ term

Solve Fibbonacci using recursion.

Write a recurrence for the same.

Is recursion advised to solve fibbonarci $n^{th}$ term?

P.S. Assuming

No addition caching like mechanism or logic in place.

$F_0 \Rightarrow 0$

$F_1 \Rightarrow 1$

$F_i \Rightarrow F_{i-1} + F_{i-2}$ , $i \geq 2$

Assignment 2:

Prove that running time of an algorithm is $\Theta(g(n))$ if and only if (iff) its worst-case running time is $O(g(n))$ & its best-case running time is $\Omega(g(n))$.