Complexity Analysis

How the alg is going to behave as the number of inputs $\to \infty$

Eg. Inventory → processes every transaction in every store in the USA.

→ 10000 ms to read the inventory from the disk.

→ 10 ms to process each transaction

n transactions ⇒

$$(10000 + 10n) \text{ ms}$$

As $n \to \infty$?

Big-Oh Notation (Upper bound)

: Abstract relationship between two functions.

Let n be the size of program input
- # of bits
- # of elements/items
- Some value.

Let $T(n)$ be a function representing running time

Let $f(n)$ be another function → profusely simple

$T(n) \in O(f(n))$ iff

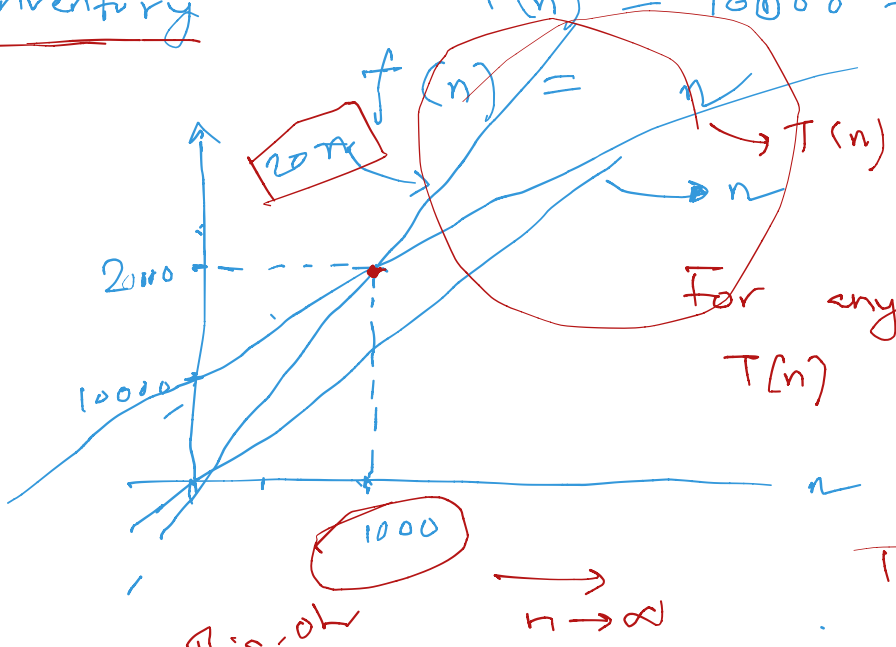
$$T(n) \leq c \cdot f(n)$$ for some const. c

whenever n is BIG for some large constant c

Big enough to make $T(n)$ fit under $c \cdot f(n)$.

$y = mx + b$

Inventory. $T(n) = 10000 + 10n$



Pick $c = 20$

$cf(n) = 20n$

for any $n >= 1000$,

$T(n) \leq c \cdot f(n)$

$\therefore T(n) \in O(f(n))$

$T(n) \in O(n)$

Big-Oh $\rightarrow n \to \infty$

Formally: $O(f(n))$ is the SET of all functions $T(n)$ that satisfies:

There exist positive constants c and N such that for all $n \geq N$, ($c \to 20$, $N \to 1000$)

$$T(n) \leq c \cdot f(n)$$

Eg. $1000000\,n \in O(n)$

Proof: Choose $c = 1000000$

$N = 0$.

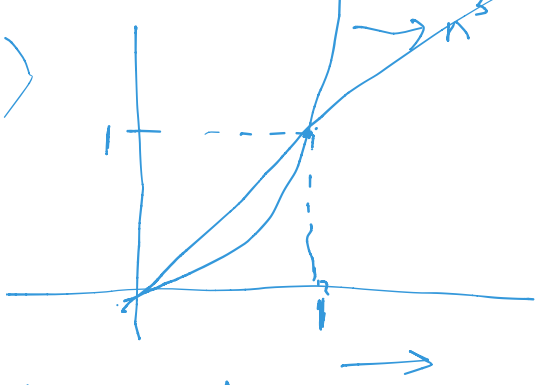
$\Rightarrow$ Big-Oh notation does not care about the constant factor.

Abstracted out algorithmic speed independent of the underlying architecture, OS, compiler.

$\therefore$ Unnecessary to write $O(2n)$ ($2 \to 1$)

(2) $n \in O(n^3)$

Proof: $c = 1$, $N = 1$



$\therefore$ Time is $\Theta(n^3)$ does not mean our algo is slow.

Big-Oh → UPPER BOUND

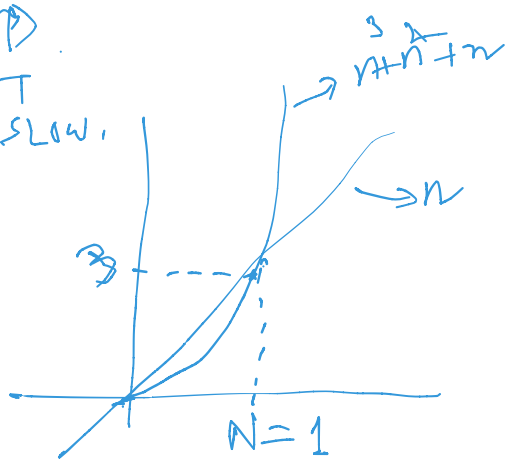
⇒ can say something is ~~FAST~~ but cannot say something is SLOW.

(3) $n^3 + n^2 + n \in O(n^3)$

Proof: $c = 3$, $N = 1$

$n^3 + n^2 + n \leq c\, n^3$

$3n^3$



Big-oh notation is usually used to indicate dominating (fastest growing) term

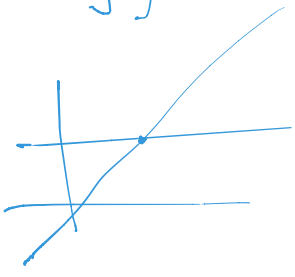


Table of important Big-oh sets (smallest to largest)

Function	Common name
$\Theta(1)$	Constant
$\subset \Theta(\log n)$	Logarithmic
$\subset \Theta(\log^2 n)$	Log-squared
$\subset \Theta(\sqrt{n})$	root-n
$\subset \Theta(n)$	Linear
$\subset \Theta(n \log n)$	n log n
$\subset \Theta(n^2)$	Quadratic
$\subset \Theta(2^n)$	Exponential

Useful

Useless

$2^n \subset e^n \subset n! \subset n^n$

$n < 100$

n →

10000 → 10^8 → # of operations

```
for ( n>{
    for(n) {
    }
}
```
→ Useless

$^{n}P_{r}$ → # of permutations. abcde...

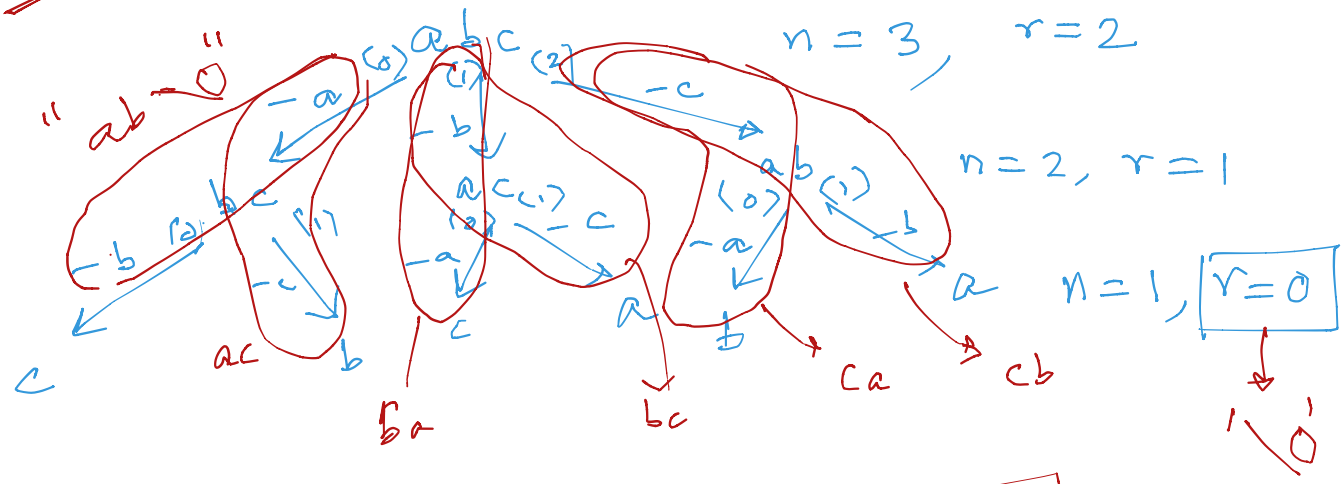
n, r

abc n=3, r=1 } a, b, c

"abc\0"

n=1, r=2 } ab, bc, ac, ba, cb, ca

output.



char* "abc\0"