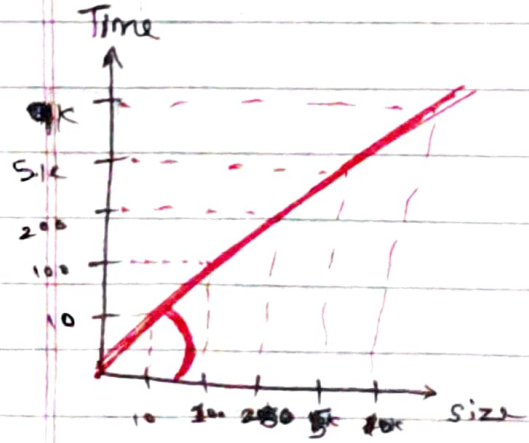
Time and Space Complexity

	Old Computer	M1 MBP (very fast)
data:	1,000,000 elements in the array	same data.
Algo:	Linear Search for target that does not exist in the array	same algorithm.
Time Taken:	10 seconds	1 second

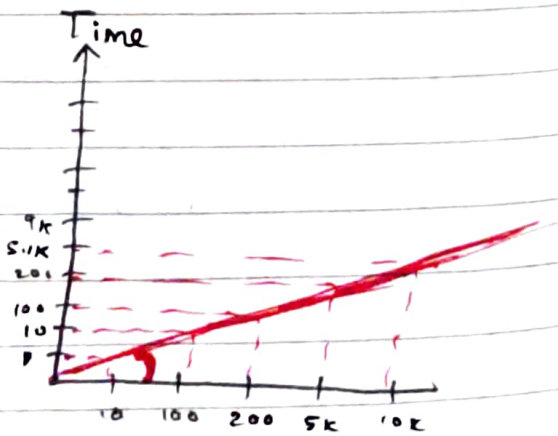
Which machine has better time complexity?
↳ Both have same time complexity.

Time Complexity ≠ Time Taken

Old machine



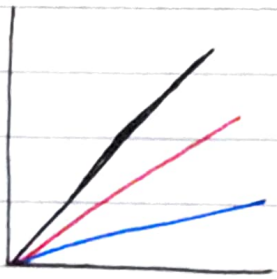
M1 MBP



Even though the time taken is different but the relationship between the size and the time is same i.e. Linear.

Time complexity is this graph, it is this mathematical function. It is a function that tells us how the time is going to grow as the input grows.

What do we need to consider when thinking about complexity?

1. Always look for worst case complexity
2. Always look at complexity for large / ∞ data
3. 

- Even though the value of actual time is different they're all growing linearly.
- We don't care about the actual time.
- Since we don't care about the actual time and only care about the relationship between the input size and time, we ignore the constants.

(4) $O(N^3 + \log N)$

From point no. (2), let N = 1 million

$O((1 \text{ million})^3 + \log(1 \text{ million}))$

$O((1 \text{ million})^3 + \underbrace{6}_{\text{very small (hence, ignored)}})$

→ Always ignore less dominating terms.

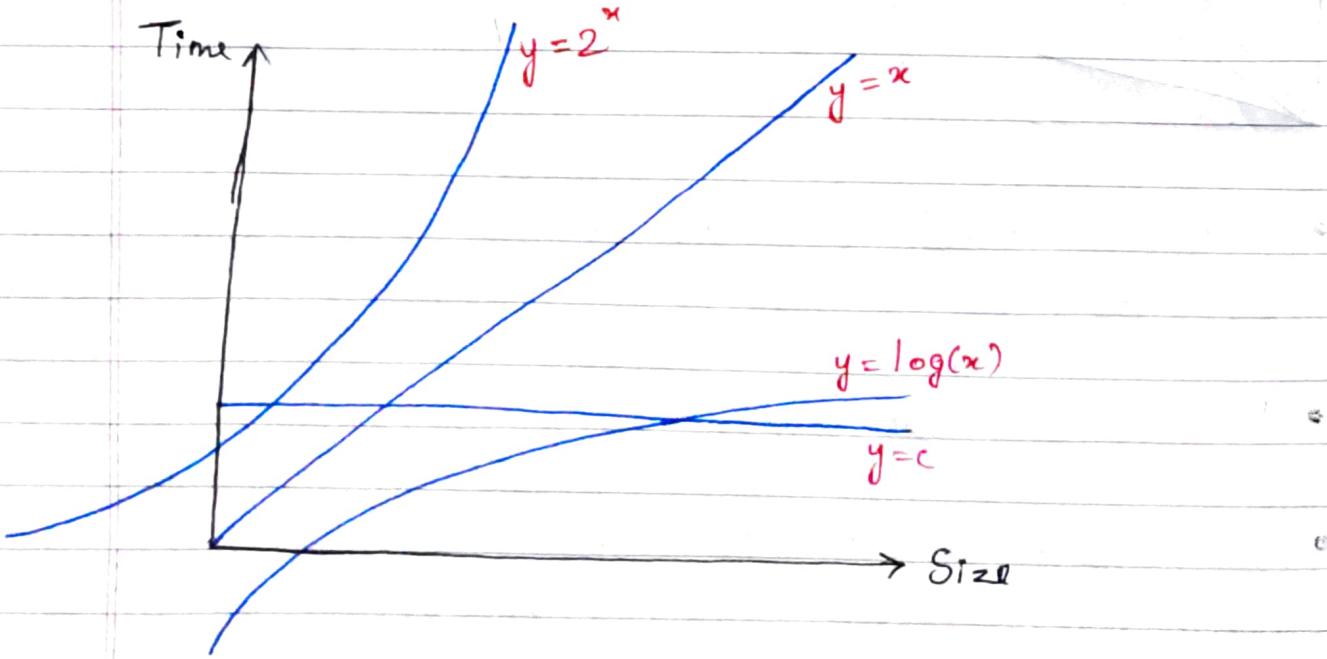
eg:- $O(3N^3 + 4N^2 + 5N + 6)$

↓ ignore all constants

$O(N^3 + N^2 + N)$

↓ ignore less dominating terms

$O(N^3)$



$O(1) < O(\log N) < O(N) < O(2^N)$

Big Oh Notation :

If we say that an algorithm has a complexity of $O(N^3)$ it means that the complexity / graph or relationship cannot exceed N^3 i.e. its the upper bound.

If $f(N) = O(g(N))$ then

$$\lim_{N \to \infty} \frac{f(N)}{g(N)} < \infty$$

eg:- $O(N^3) = O(6N^3 + 3N + 5)$

$g(N)$ $\quad$ $f(N)$

$$\lim_{N \to \infty} \frac{6N^3 + 3N + 5}{N^3}$$

$$\lim_{N \to \infty} 6 + \frac{3}{N^2} + \frac{5}{N^3} \rightarrow 6 + \frac{3}{\infty^2} + \frac{5}{\infty^3} \rightarrow 6$$

finite Value $\leftarrow$ $(6 < \infty)$

Big Omega Notation :

$\Omega(N^3)$ $\rightarrow$ lower bound (It will never be less than this)

$$\lim_{N \to \infty} \frac{f(N)}{g(N)} > 0$$

Theta Notation :

$\Theta(N^2)$ => Both, upper and lower bound is N^2

$$0 < \lim_{N \to \infty} \frac{f(N)}{g(N)} < \infty$$

Little Oh Notation :

This is also going to give upper bound but this is not strict upper bound. It is a loose upper bound.

Big Oh	Little Oh (Stronger statement)
~~$f \neq O(g)$~~	~~$f \neq o(g)$~~
$f = O(g)$	$f = o(g)$
$f \leq g$	$f < g$ (Strictly)

$$\lim_{N \to \infty} \frac{f(N)}{g(N)} = 0$$

eg: $f = N^2$ $\quad g = N^3$

$$\lim_{N \to \infty} \frac{N^2}{N^3} = \lim_{N \to \infty} \frac{1}{N} = 0$$

Little Omega Notation:

This is also going to give lower bound but this is not strict lower bound It is a loose lower bound.

Big Omega	Little Omega
$f = \Omega(g)$	$f = \omega(g)$
$f \geqslant g$	$f > g$

$$\lim_{N \to \infty} \frac{f(N)}{g(N)} = \infty$$

Space Complexity or Auxiliary Space

Auxiliary Space is the extra space or temporary space used by an algorithm.

Space Complexity of an algorithm is total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary Space and the space used by the input.

For example, if we want to compare standard sorting algorithms on the basis of space, then Auxiliary Space would be a better criteria than Space Complexity. Merge Sort uses O(N) auxiliary space, Insertion sort, Heap sort use O(1) auxiliary space. Space complexity of all these sorting algorithms is O(N) though.

Q:

```
for (i=1; i≤N;) {
    for (j=1; j≤k; j++) {
        // some operation
           that takes t time
    }
    i = i + k
}
```

Find the time complexity

Inner loop : O(kt) time

Outer loop : O(Kt * times outer loop is running)

$i = 1, 1+K, 1+2K, 1+3K, 1+4K, \ldots 1+xK$

$1 + xK \leq N$

$xK \leq N-1$

$x \leq \frac{N-1}{K}$

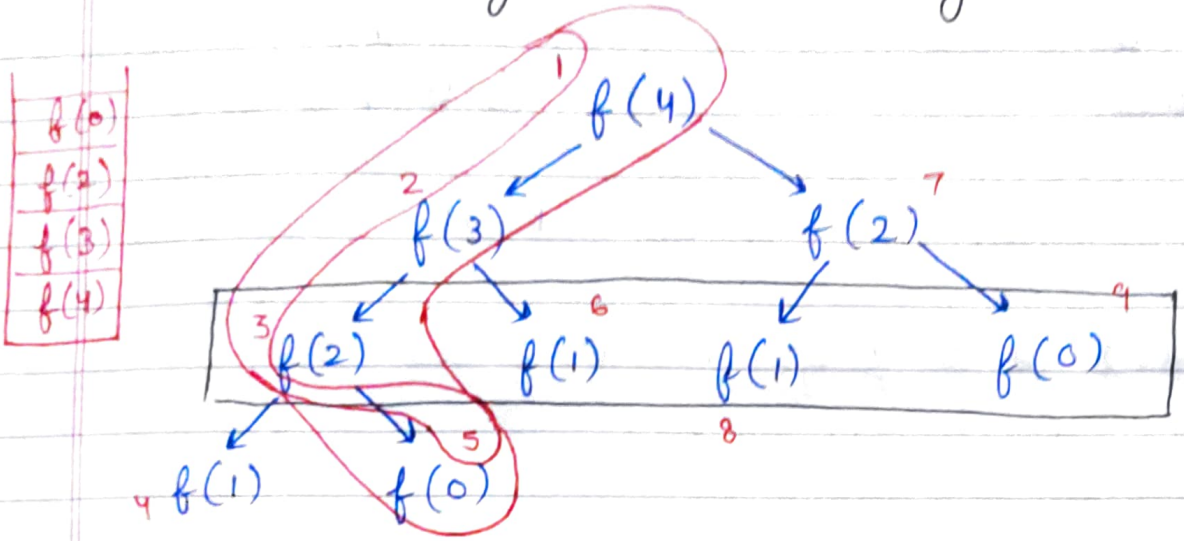
$x = \frac{N-1}{K}$

$O\left(Kt * \frac{N-1}{K}\right)$

$O(t*(N-1))$

$O(Nt)$

Recursive Algorithms Analysis



☆ No two function calls at the same level of recursion will be in the stack at the same time.

☆ Only calls that are interlinked with each other will be in the stack at the same time.

So, what is the maximum space that we will be taking? Is it possible that all the nine function calls will be in the stack at the same time? → NO

The longest chain running from the root to leaf node is the maximum amount of space we'll be taking → The height of the tree

Space Complexity = The height of the tree

O(N)

Recurrences

Linear

$F(N) = F(N-1) + F(N-2)$

Divide & Conquer

$F(N) = F\left(\frac{N}{2}\right) + O(1)$

Divide & Conquer Recurrences :

Form :

$$T(x) = a_1 T(b_1 x + \varepsilon_1(x)) + a_2 T(b_2 x + \varepsilon_2(x)) + \cdots + a_k T(b_k x + \varepsilon_k(x)) + g(x)$$

$\forall \; x \geq x_0$ ← some constant

$$T(N) = T\left(\frac{N}{2}\right) + C \quad \left\{ a_1 = 1,\ b_1 = \frac{1}{2},\ \varepsilon_1(x) = 0,\ g(x) = c \right.$$

$$T(N) = 9\,T\left(\frac{N}{3}\right) + \frac{4}{3}\,T\left(\frac{5N}{6}\right) + \underbrace{4N^3}_{g(N)}$$

(a_1, b_1, a_2, b_2)

$$T(N) = 2\,T\left(\frac{N}{2}\right) + (N-1)$$

when you get ans from this + what you're doing takes how much time

How to actually solve recurrences to get complexity.

① Plug & Chuck $\quad F(N) = F\left(\frac{N}{2}\right) + C$

putting $N \to \frac{N}{2}$, and then you know the drill.

② Master's Theorem (Ignore this theorem)

③ **AKra Bazzi** (Best method)

$$T(x) = \Theta\left(x^p + x^p \int_1^x \frac{g(u)}{u^{p+1}}\, du\right)$$

$$a_1 b_1^p + a_2 b_2^p + \cdots\cdots = 1$$

$$\sum_{i=1}^{k} a_i b_i^p = 1$$

Q $\quad T(N) = 2T\left(\frac{N}{2}\right) + (N-1)$

$a_1 = 2, \; b_1 = \frac{1}{2}, \; g(x) = x - 1$

$$2 \cdot \left(\frac{1}{2}\right)^p = 1 \longrightarrow p = 1$$

$$T(x) = \Theta\left(x^1 + x^1 \int_1^x \frac{u-1}{u^{1+1}}\, du\right)$$

$$T(x) = \Theta\left(x + x\int_1^x \frac{u-1}{u^2}\,du\right)$$

$$T(x) = \Theta\left(x + x\int_1^x \frac{du}{u} - x\int_1^x \frac{du}{u^2}\right)$$

$$T(x) = \Theta\left(x + x\left[\log(u)\right]_1^x + (-x)\left[-u^{-1}\right]_1^x\right)$$

$$= \Theta\left(x + x\log x + (-x)\left(1 - \frac{1}{x}\right)\right)$$

$$= \Theta\left(x + x\log x - x + 1\right)$$

$$= \Theta\left(x\log x + 1\right)$$

$$\boxed{T(x) = \Theta\left(x\log x\right)}$$ → for array of size N → $O(N \log N)$

Q $$T(N) = 2T\left(\frac{N}{2}\right) + \frac{8}{9}T\left(\frac{3N}{4}\right) + N^2$$

$a_1 = 2$, $b_1 = \frac{1}{2}$, $a_2 = \frac{8}{9}$, $b_2 = \frac{3}{4}$, $g(u) = u^2$

$$2\cdot\left(\frac{1}{2}\right)^p + \frac{8}{9}\cdot\left(\frac{3}{4}\right)^p = 1 \longrightarrow p = 2$$

$$T(x) = \Theta\left(x^2 + x^2\int_1^x \frac{u^2}{u^3}\,du\right)$$

$$T(x) = \Theta\left(x^2 + x^2 \int_1^x \frac{du}{u}\right)$$

$$= \Theta\left(x^2 + x^2 [\log(u)]_1^x\right)$$

$$= \Theta\left(x^2 + x^2 \log x\right)$$

$$= \Theta\left(x^2 \log x\right)$$

If you can't find the value of p :-

$$T(x) = 3T\left(\frac{x}{3}\right) + 4T\left(\frac{x}{4}\right) + x^2$$

Let's try $p=1$

$$3 * \left(\frac{1}{3}\right)^1 + 4 \cdot \left(\frac{1}{4}\right)^1 \neq 1$$

$2 > 1 \rightarrow$ Increase the denominator

$\therefore$ $p > 1$

try $p=2$

$$3 \cdot \left(\frac{1}{3}\right)^2 + 4 \cdot \left(\frac{1}{4}\right)^2 \neq 1$$

$$\frac{1}{3} + \frac{1}{4}$$

$$\frac{7}{12} < 1$$

$\therefore$ $p < 2$

NOTE: When $p <$ power of $g(x)$ then answer $= g(x)$

here, $g(x) = x^2$ and $p < 2$ (i.e power of $g(x)$)

hence, ans $= O(g(x))$

Why ?

$$T(x) = \Theta\left(x^p + x^p \int_1^x \frac{u^2}{u^{p+1}}\, du\right)$$

$$\Theta\left(x^p + x^p \int_1^x u^{1-p}\, du\right)$$

$$\Theta\left(x^p + x^p \left[\frac{u^{1-p+1}}{1-p+1}\right]_1^x\right)$$

$$\Theta\left(x^p + x^p \left(x^{2-p} - 1\right).\frac{1}{2-p}\right)$$

$$\Theta\left(x^p + x^2\right)$$

as we know that $p<2$, so x^p is ignored

$$T(x) = \Theta(x^2)$$

$$T(x) = \Theta(g(x))$$

Solving Linear Recurrences (Homogenous)

Form: example → $F(N-1) + F(N-2) = F(N)$

$$f(x) = a_1 f(x-1) + a_2 f(x-2) + a_3 f(x-3) + \cdots + a_n f(x-n)$$

$$f(x) = \sum_{i=1}^{n} a_i f(x-i) \quad \forall\ a_i,\ n \text{ is fixed}$$

$n \rightarrow$ order of recurrence

Solution for fibonacci numbers :-

$$f(n) = f(n-1) + f(n-2) \quad \text{—— (1)}$$

(1) → Put $f(n) = \alpha^n$ for some constant α

$$\alpha^n = \alpha^{n-1} + \alpha^{n-2}$$

$$\alpha^n - \alpha^{n-1} - \alpha^{n-2} = 0$$ → Characteristic equation of recurrence

$$\alpha^2 - \alpha - 1 = 0 \quad [\text{divided by } \alpha^{n-2} \text{ both sides}]$$

roots of this eqⁿ

roots of quadratic → $\left[\because \frac{-b \pm \sqrt{b^2-4ac}}{2a}\right]$

$$\alpha = \frac{1 \pm \sqrt{5}}{2}$$

$$\alpha_1 = \frac{1+\sqrt{5}}{2}, \quad \alpha_2 = \frac{1-\sqrt{5}}{2}$$

(2) If you have α_1 and α_2 as two roots then you can write the equation as

$$f(n) = c_1\alpha_1^n + c_2\alpha_2^n$$ is a solution for fibonacci

$$f(n) = c_1\left(\frac{1+\sqrt{5}}{2}\right)^n + c_2\left(\frac{1-\sqrt{5}}{2}\right)^n \quad -(2)$$

(3) FACT: No. of roots that you have = No. of answers that you have already

here, we have two roots, hence, we should have two answers already and we do have them,

$f(0) = 0$ and $f(1) = 1$

$$f(0) = 0 = c_1 + c_2 \Rightarrow c_1 = -c_2 \quad -(3)$$

$$1 = c_1\left(\frac{1+\sqrt{5}}{2}\right) - c_1\left(\frac{1-\sqrt{5}}{2}\right) \quad [\text{from } (3)]$$

$$c_1 = \frac{1}{\sqrt{5}} \quad \& \quad c_2 = -\frac{1}{\sqrt{5}}$$

putting the values of c_1 & c_2 in (2)

$$f(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

→ formula for n^{th} fibonacci number

$$f(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

$$= \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

as $n \to \infty$, this term will become close to zero. Hence, it is the less dominating term and will be ignored.

Time Complexity

$$O\left(\frac{1+\sqrt{5}}{2}\right)^n$$

$$O(1.6180)^n$$

golden ratio

$$T(N) = O(1.618)^n$$

Solving Recurrence Relations with Repeated Roots

$$f(n) = 2f(n-1) - f(n-2)$$

(1) Let $f(n) = \alpha^n$

$$\alpha^n = 2\alpha^{n-1} - \alpha^{n-2}$$

$$\alpha^n - 2\alpha^{n-1} + \alpha^{n-2} = 0$$

$$\alpha^2 - 2\alpha + 1 = 0 \longrightarrow (\alpha = 1)$$ double root

General Rule → If α is repeated 'r' times then $\alpha, n\alpha^n, n^2\alpha^n, \ldots$ $\ldots, n^{r-1}\alpha^n$ are all solutions to the recurrence

here, we can take two roots → $1, n\alpha^n$

$$f(n) = c_1(\alpha)^n + c_2(n\alpha^n)$$

$$f(n) = c_1 + nc_2$$

If, $f(0) = 0$ & $f(1) = 1$ then

$$f(0) = c_1 = 0 \quad \because f(0) = 0$$

$$f(1) = c_1 + c_2 = 1$$

$$c_2 = 1$$

$$f(n) = c_1 + nc_2$$

$$f(n) = n \longrightarrow O(n)$$

Solving Non-Homogenous Linear Recurrences:

$$f(n) = a_1 f(x-1) + a_2 f(x-2) + a_3 f(x-3) + \cdots + a_d f(x-d) + g(x)$$

when this extra function is there, it is known as non-homogenous linear recurrence

Steps:

① Replace $g(x)$ by 0 & solve usually

$f(n) = 4f(n-1) + 3^n$, $f(1) = 1$

replacing 3^n with 0,

$f(n) = 4f(n-1)$

$\alpha^n = 4\alpha^{n-1}$

$\alpha^n - 4\alpha^{n-1} = 0$

$\alpha - 4 = 0$

$\alpha = 4$

Homogenous sol^n → $f(n) = c_1\alpha^n$ → $c_1 4^n$

(2) Take $g(n)$ on one side and find particular solution.

$$f(n) - 4f(n-1) = 3^n$$

Now, guess something that is similar to $g(n)$

If $g(n) = n^2$ then guess a polynomial of degree 2.

my guess: $f(n) = c3^n$

$$c3^n - 4c3^{n-1} = 3^n$$

$$3c - 4c = 3$$

$$c = -3$$

particular sol^n $\Rightarrow$ $f(n) = -3^{n+1}$

(3) Add both solutions together.

$$f(n) = c_1 4^n + (-3^{n+1})$$

we know that, $f(1) = 1$

$$(-3^{(1+1)}) + 4c_1 = 1$$

$$4c_1 = 1 + 9$$

$$c_1 = \frac{5}{2}$$

$$f(n) = \frac{5}{2} \cdot 4^n - 3^{n+1}$$

How do we guess the particular sol^n?

★ If $g(n)$ is exponential, guess of the same type.

Ex: $g(n) = 2^n + 3^n$

guess: $f(n) = a\,2^n + b\,3^n$

★ If $g(n)$ is polynomial, guess of the same degree.

Ex: $g(n) = n^2 - 1$

guess: $f(n) = an^2 + bn + c$

★ If it is polynomial and exponential both then:

Ex: $g(n) = 2^n + n$ guess: $a\,2^n + (bn + c)$

Let's say you guessed, $f(n) = a\,2^n$ and it fails, then try $(an+b)\,2^n$, if this also fails then increase the degree, $(an^2+bn+c)\,2^n$

Ex: $f(n) = 2f(n-1) + 2^n$, $f(0) = 1$

replace 2^n with 0,

$$f(n) = 2f(n-1)$$

$$\alpha^n = 2\alpha^{n-1}$$

$$\alpha^n - 2\alpha^{n-1} = 0$$

$$\alpha - 2 = 0 \rightarrow \alpha = 2$$

guess particular sol^n $\rightarrow f(n) = a\,2^n$

$$a2^n - 2a2^{n-1} = 2^n$$

$$a = a + 1$$ ✗ wrong

hence, guess another one from our rule,

$$f(n) = (an+b)\,2^n$$

$$(an+b)2^n = 2(a(n-1)+b)2^{n-1} + 2^n$$

$$an + b = an - a + b + 1$$

$a = 1$ discard b

$f(n) = n2^n \rightarrow$ particular sol^n

general ans :→ $f(n) = c_1 2^n + n2^n$

$f(0) = 1 = c_1 + 0 \quad , \quad c_1 = 1$

$$f(n) = 2^n + n2^n$$

complexity $= O(n2^n)$