Análisis de Algoritmos

Puntos que tomar en cuenta al diseñar un algoritmo o programa:

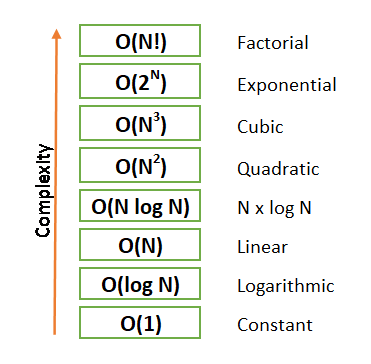
1. Modularidad de código
2. Que esté bien documentado.
3. Que este correcto el código (saber probarlo etc…)
4. Eficiencia del código (utilizar la mínima cantidad posible de recursos)
   1. tiempo (número de operaciones básicas)
   2. espacio (memoria)
   3. comunicación: menos mensajes intercambiados posibles

Running time vs memory: Generalmente, se prioriza el tiempo sobre la memoria. El tiempo acota.

Quiz: ¿Qué es más eficiente, imprimir cada uno de los elementos de un arreglo con system out, o agregar todos los elementos a 1 string y solo imprimir ese string?

**Best, Worst and Average Case Scenario**

* Mejor de los casos: De poco uso práctico, casi nunca pasa, pero que belleza y qué gloria cuando si.
* Caso Promedio: Puede ser útil porque es como funcionara el programa la mayoría del tiempo, pero es difícil de saber
* Peor de los casos: La medida de performance más usada. Ciertamente, cuando mas ocupas el programa es cuando mejor debe de rendir (e.g. piensa en el avión autopiloto colapsando)
* Caso Amortizado: ie, usar la estructuras de datos como se debe nos garantiza que siempre será mejor que el peor de los casos

Si medimos la inserción de un arreglo por el peor de los casos, ¿no son los arreglos terribles? Cada vez que quieras insertar en un arreglo lleno, tienes que expandir y luego insertar (ie, mover toooodos los datos del arreglo a uno nuevo, y solo después, insertar el dato nuevo), mientras que la inserción en una lista es siempre al final.

Pues no, porque el array tiene un modo de operación y la lista otro. Las estructuras de datos tienen una función específica. No es lo mismo querer borrar el primer elemento en un Stack que en un Queue, y sería absurdo decir que un Queue es más eficiente porque tarda menos en eliminar el primer dato .

El arreglo como una ED:

* Insertar/borrar: una sola operación. El peor de los casos es que el arreglo está lleno y llama a una operación llamada *expand*: aquí se crean N nuevas direcciones de memoria (total de 2N), luego se cambian las N direcciones de memoria al nuevo arreglo y finalmente se insertará lo que deseas.

**Notación Big O**

El factor que determina la eficiencia de un algoritmo es el número de pasos que toma. No podemos, sin embargo, decir que un algoritmo es de “22 pasos” y otro de “400 pasos” ¿Por qué? Porque la gran mayoría de las veces, los algoritmos varían dependiendo del número de datos que tengamos.

Una definición rudimentaria de la Notación Big O sería: Si se tienen N elementos, ¿Cuántos pasos tomará el algoritmo?

Por ejemplo, un ciclo for que imprime todos los elementos de un arreglo de n elementos se denota O(N), y el programa “Hola Mundo” es O(1).

Pero, si tu algoritmo tiene siempre 3 pasos, ¿Cuál es su notación Big O?

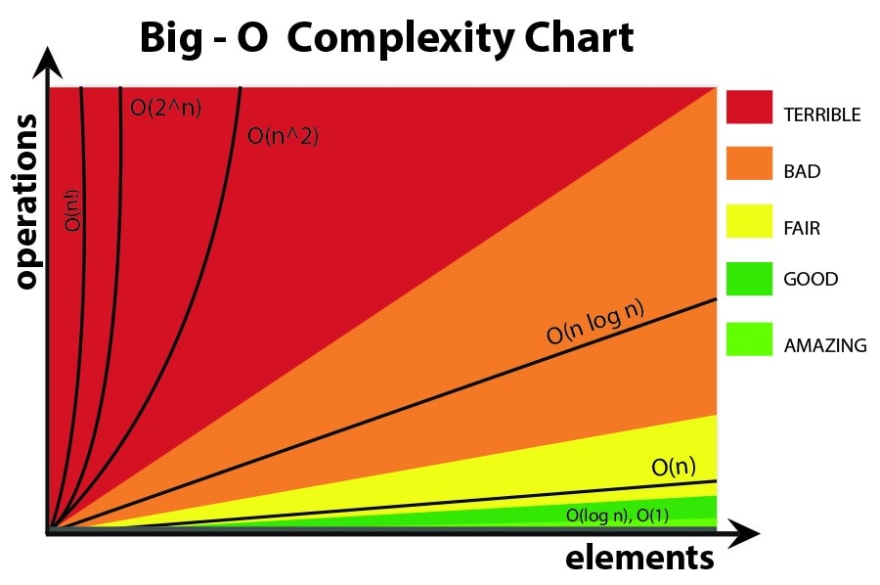
1. O(3)
2. O(1)

De hecho, es O(1)! ¿Por qué, si tiene 3 pasos?

Esto es el alma de Big O: No te dice solamente cuantos pasos toma un algoritmo, sino como aumentan los pasos conforme aumenta el número de datos. Esa es la verdadera definición.

Esto puede ser confuso, porque por ejemplo, O(100N) es, en términos de Big O, lo mismo que O(2N). ¡Y eso que O(100N) es 50 veces más lento que O(2N)!

Para entender este concepto que a primera vez pudiera parecer absurdo, hay que saber que Big O se concentra con categorías generales.



Por ejemplo, para comparar una casa y un rascacielos, no decimos “la casa de 2 pisos vs. el rascacielos de 80 pisos”, simplemente decimos “la casa vs el rascacielos”; se vuelve absurdo especificar los pisos entre edificios tan diferentes.

Del mismo modo, Big O se usa para denotar cómo crece la trayectoria de un algoritmo entre más incrementen los datos.

O(100N), O(2N) y O(N) crecen linealmente.

O(N^2 / 100), O(31\*N^2) y O(N^2) crecen exponencialmente

Las tasas de crecimiento son totalmente diferentes. Si comparamos un algoritmo lineal con un algoritmo cuadrático (Por ejemplo, O(N^2 / 2) vs O(100N)), especificar que el cuadrático está dividido entre 2 y que el lineal está multiplicado por 100 es como especificar el número de pisos cuando comparamos una casa con un rascacielos.

Pero si comparamos a 2 algoritmos de la misma categoría (e.g. lineal vs lineal), no significa que tengan la misma velocidad.

Notación asintótica: O(g(n))={f(n) existen constantes positivas c0 y n0 tal que 0 <= f(n) <= cg(n) para toda n>=n0}

**Categorias de Big O y casos Comunes**:

Casos Bonitos

* O(1): Constante

Algoritmos con número de pasos independiente del número de datos. e.g. Imprimir “Hola mundo”, imprimir el primer elemento de un array

* O(log N): Logaritmo

Algoritmos cuyos pasos aumentan cada vez que se doblan los datos. e.g. Búsqueda Binaria.

* O(N): Lineal

Algoritmos cuyos pasos aumentan al mismo ritmo que aumentan los datos. e.g. Imprimir todos los datos de un array

* O(n log N):

Algoritmo cuyos pasos aumentan cada cierto numero de datos. Dependiendo del número de datos a manejar, puede ser mas lento (si son pocos) o mas rapido (si son muchos) que O(N).

Casos Feos

* O(N^2): Cuadrático

Algoritmos cuyos pasos aumentan por n por cada dato extra. Pasa mucho con nested loops.

* O(C^N): Exponencial

Pasa muchísimo con recursión dentro de otras recursiones.

* O(N!): Factorial

Algoritmos cuyos pasos aumentan de manera factorial; el peor de los peores casos. Pasa mucho cuando haces combinaciones y permutaciones (e.g. Encontrar todos los palindromos, o el problema del vendedor viajero)

**Notación Big - Ω**

A veces queremos decir que un algoritmo toma, por lo menos, una cierta cantidad de tiempo. Para eso se usa Big Omega.

**Ω(g(n))**

Esta notación es similar a la de Big O, solo que se acota la función g(n) por abajo, utilizando el mismo valor de n0.

**Θ(g(n))**

Cuando Ω(g(n)) = O(g(n)) se utiliza la notación Big-Θ

Diferencia entre Big O y Big Omega: Big O describe el límite/cota superior, Big Omega describe el límite/cota inferior

Complejidad

Lo que se tarda, número de pasos o recursos utilizados para que el código más eficiente corra y solucione el problema.

Tiempo Polinomial

Función polinomial de complejidad temporal en la notación Big - O de forma O(p(n))

Hay problemas para los cuales AUN no existen algoritmos para solucionarlo en tiempo polinomial. A estos problemas se les describe como NP (No determinista en tiempo polinómico)

Hay problemas que son tan difíciles como el algoritmo para resolverlo, en este caso, los problemas son NP-Completos. Son los problemas más alejados del tiempo polinomial

P=NP ?

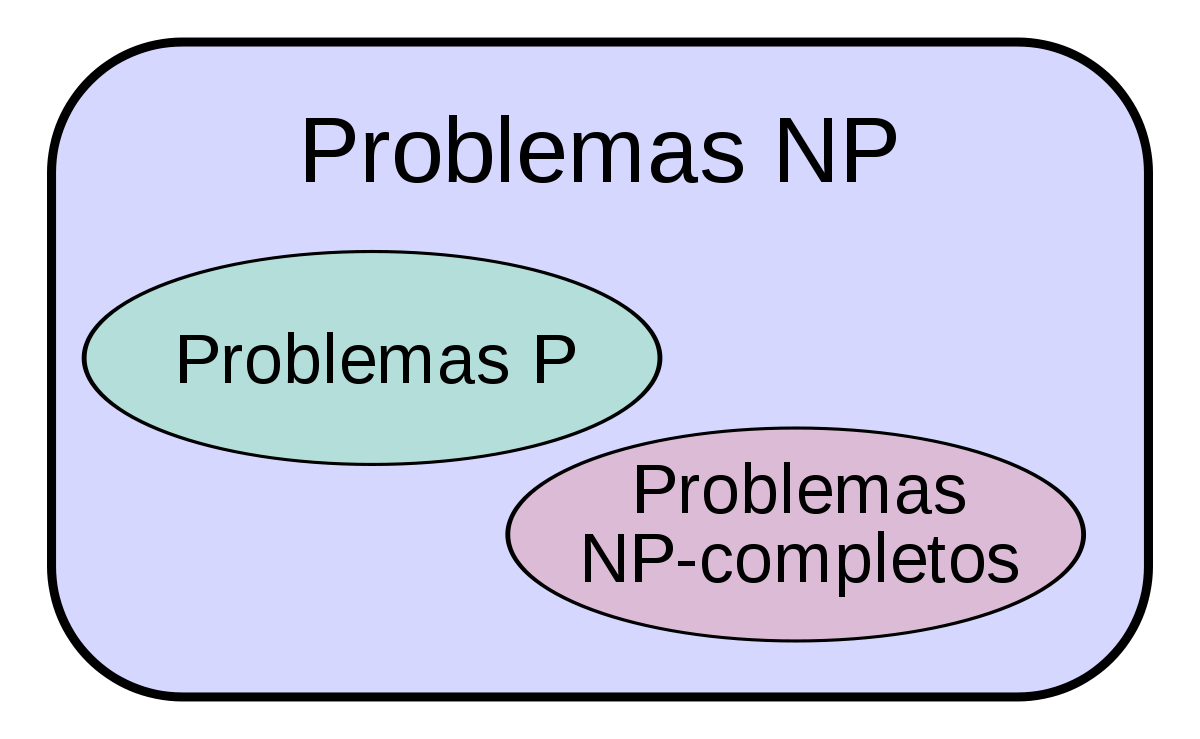
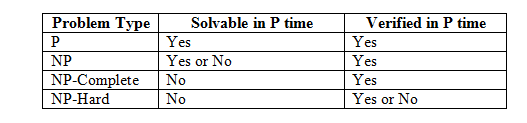
Si P=NP, significa que existen algoritmos para solucionar los problemas de NP en tiempo polinomial.

Si P≠NP, significa que existen problemas que no tienen una solución en tiempo polinomial.

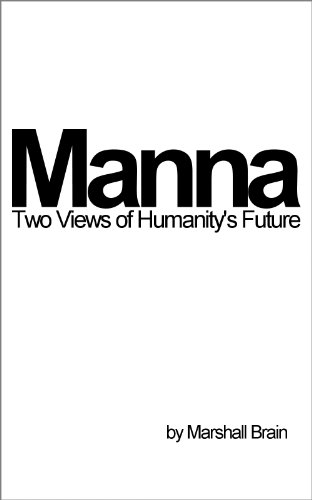
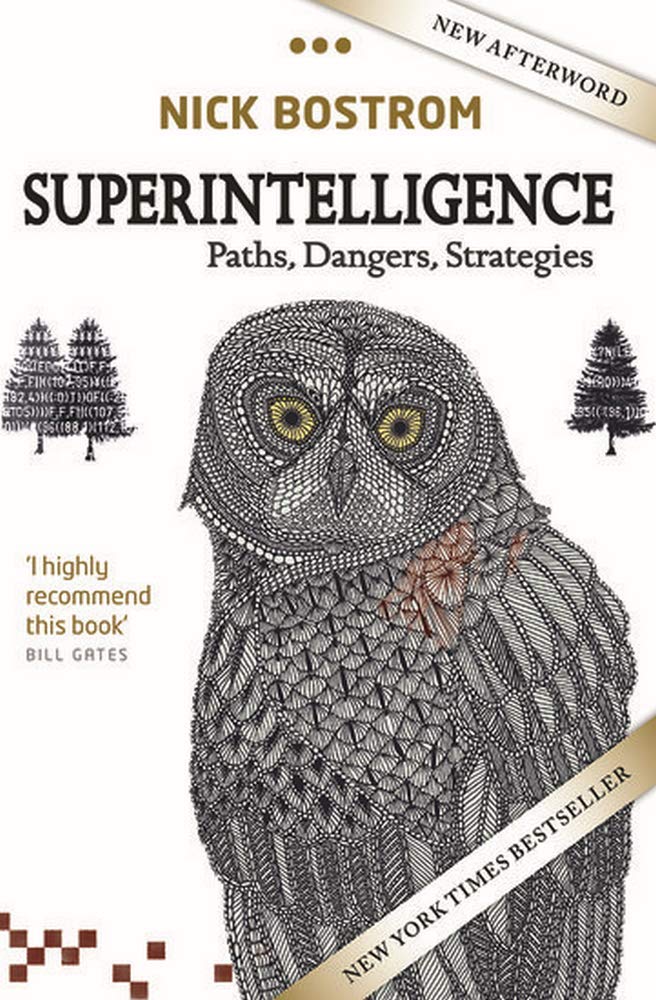
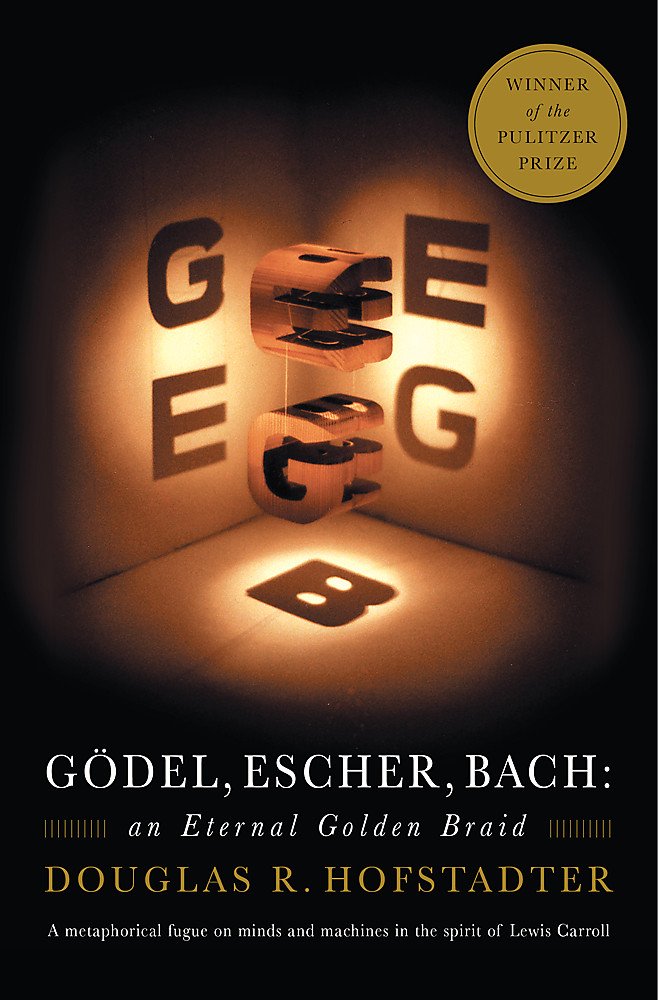
(muuuuy problablemente no, son fumaderas pensar que si, se podría colapsar la realidad como la conocemos si realmente fuera el caso que todos los problemas tienen solución fácil y existirían cosas como el viaje con mayor velocidad que la luz, el Oraculo/Demonio de Laplace y cómo implementar el comunismo espacial de lujo totalmente automatizado.)

*“Proving that P does not equal NP would just confirm what almost everyone already assumes to be true for practical purposes,” explains Scott Aaronson*

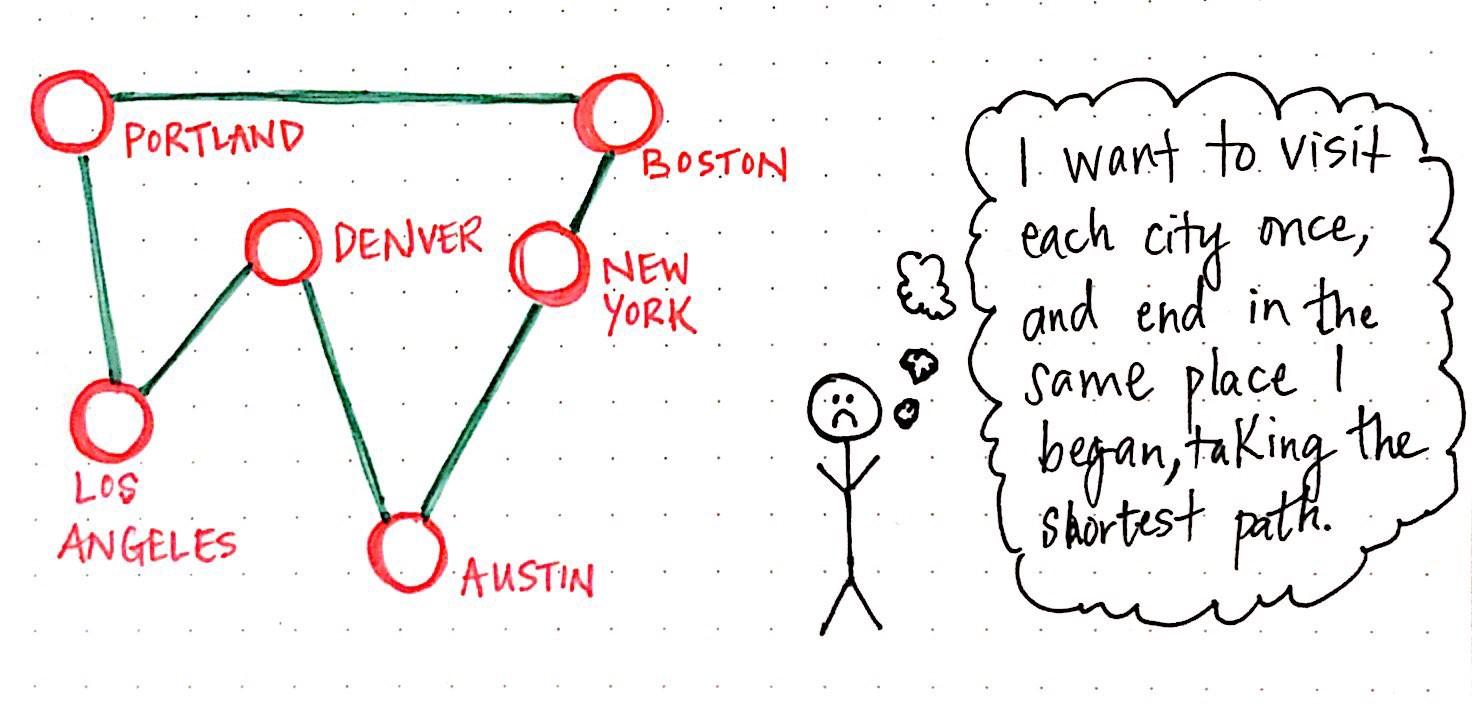
<https://www.technologyreview.com/2010/08/19/262224/what-does-p-vs-np-mean-for-the-rest-of-us/> )



**Deep Lore pa Ideas 4 y gente ociosa**



**El Vendedor Viajero**

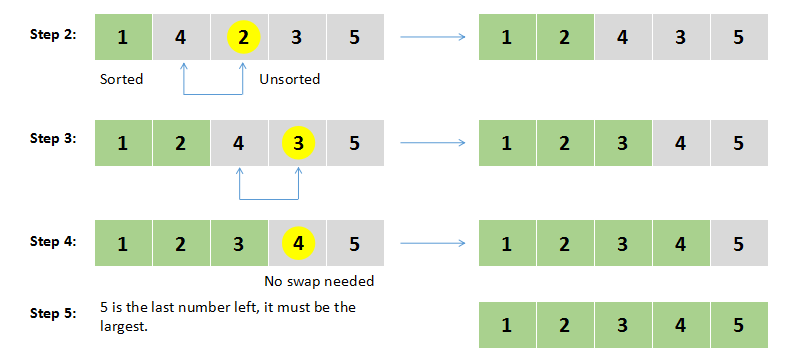


Imagina que tienes 4 ciudades y sus distancias en kilómetros, y quieres viajar a todas las ciudades exactamente 1 vez, empezando y terminando en la misma ciudad. ¿Cómo eliges la mejor ruta?

Hasta ahora, el único modo es escribir todas las rutas posibles y comparar las distancias. El problema es que es terriblemente ineficiente entre más puntos/ciudades tengas. Por ejemplo, con 30 puntos/ciudades, el número de caminos posibles es 29!=8.84\*10^30, y si pudieras calcular cada circuito en 1 nanosegundo, te tomaría 28,000,000,000,000,000,000 años descifrar todos los caminos. ¡Y luego tienes que compararlos!

En este sentido, las cadenas de logística modernas son nada menos que una maravilla. ¡Por eso les pagan tanto a los Científicos de Datos de Amazon!

**Selección Directa/Selection Sort**

El algoritmo de selección directa encontrará el elemento desordenado más pequeño en cada iteración, y lo pondrá al final de los datos ordenados.

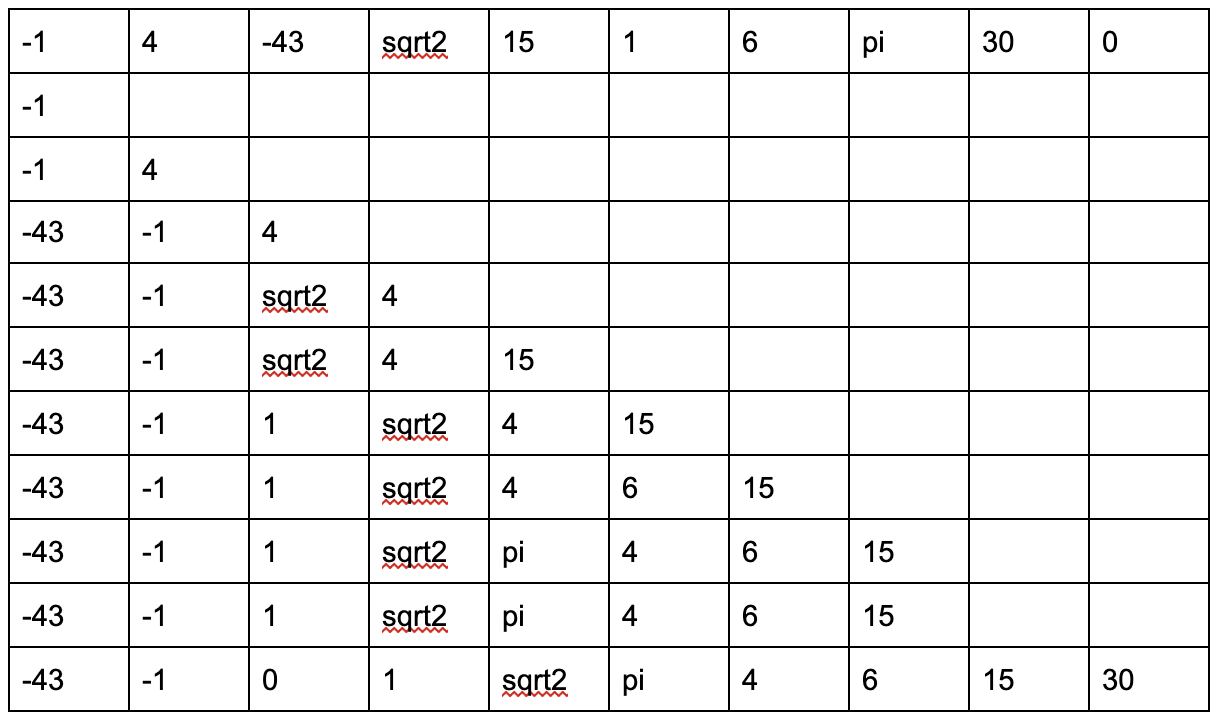
Si consideramos que este algoritmo usa 2 ciclos, y en el peor de los casos ocupa intercambiar datos en cada iteración por que los datos están ordenados de mayor a menor, su eficiencia en términos de Big O es O(N^2). [Mejorar la explicacion]

**Insertion/Quick Sort**

Selection Sort aplicaría N^2 por la N del for, lo que implicaría N^3 lo cual se complica de más.

Mejor se usa Insertion Sort, la cual recibe el arreglo con los datos.

Para evitar comparar con todos, se crea una variable “límite”, la cual permite comparar solamente con el anterior, y en el caso de que sea mas chico que el anterior, los intercambia y hace lo mismo con el siguiente, hasta que sea mayor y ahí se queda. Como se usa un while dentro de un for, se hace N^2.



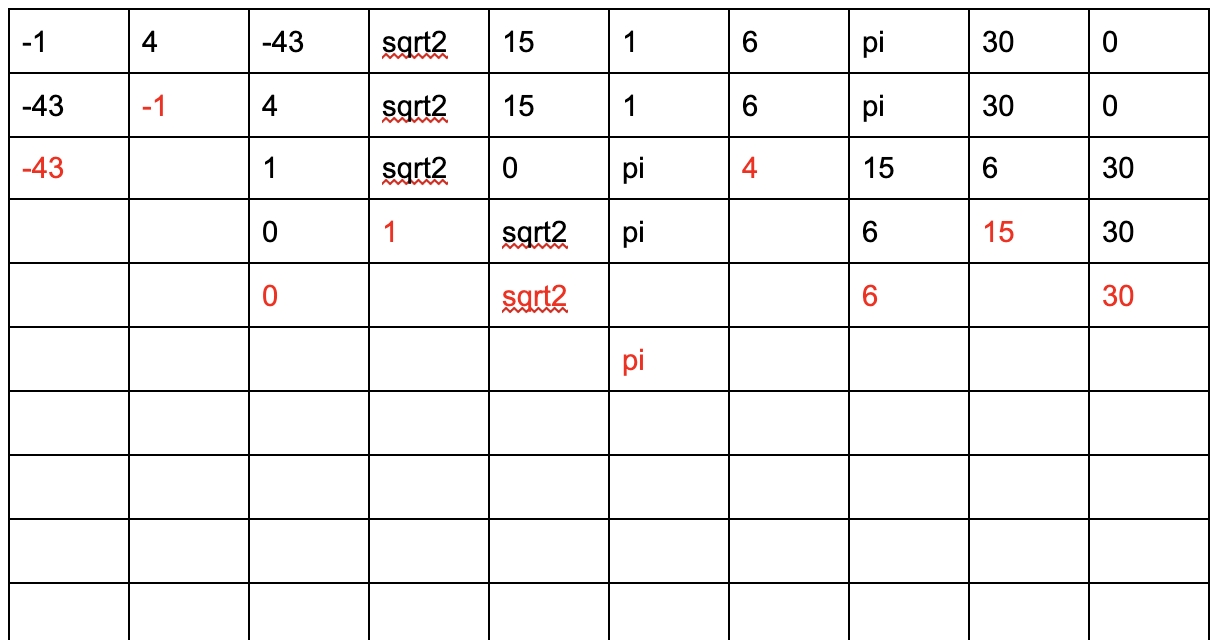
Quick sort, recibe los datos, “

Se hace una partición, se analiza el arreglo para después elegir un pivote y se coloca en la posición correcta. Los datos más chicos se ponen a la izquierda del pivote, y los mayores a la derecha, generando cierta idea de cómo se podrían ordenar los demás datos.

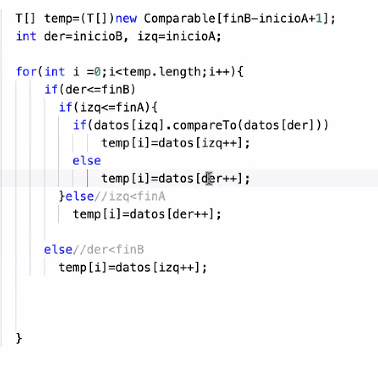
Método partición(inf,sup,datos)

dos índices, uno sube y uno baja, Cuando compara y es más grande que el pivote, lo pone en la posición del índice que baja, y si es mas chico en el índice que sube, al finalizar el método, los más grandes estarán a la derecha del pivote y los chicos a la izquierda, ya que los índices se igualan en la posición del pivote. (Esta es solo una de las posibles maneras de hacer este método)

Tip: el Quick Sort----- Qs(int,pivote,datos) lo hace n\* lg(n), mucho más rápido.



clase 9/6/2021



se checa que der e izq esten dentro del limite, se mete el más chiquito y se avanza.



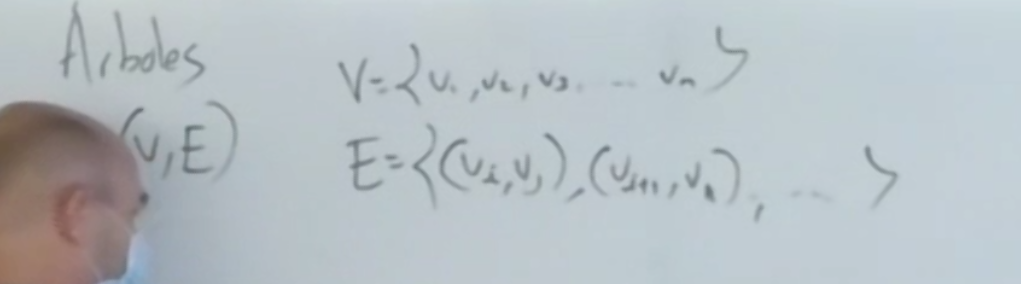
nota: estos algoritmos tienen una pequeña condición. Aquí suponemos que los datos que queremos ordenar caben en un arreglo pero no siempre será así.

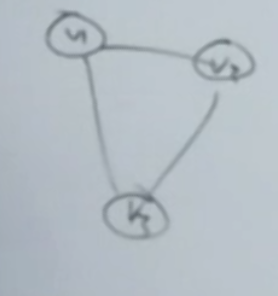
¿cómo le hago para ordenar cosas que no caben en su memoria principal?

cinta =acceso secuencial, más baratas que un disco duro.

arbol puede verse como grafo

es una pareja ordenada de vertices y aristas (e)

esto es def matematica

el grafo  conjunto de esto, es la bolita y el palito.

lo podemos ver como:

bolitas: alumnos

palitos: amigos o sea bolita 1 con bolita 2 son amigos

el conjunto vacio es un arbol

con nexo :existe un camino entre cualquier pareja de vertices

aciclico: no ciclico, no existe camino que pase por el mismo nodo dos veces sin repetir arista