Algoritmos de búsqueda y ordenamiento

# ¿Qué es un algoritmo?

## Definición

* Es un conjunto de instrucciones precisas y finitas que al ejecutarse permiten resolver un problema.
* No deben tener ambigüedad (Debe ser claro) y debe ser eficaz. No necesariamente eficiente, es solo un valor ideal.
* Es finito porque termina después de ejecutar todos los pasos.
* Posee un input o un output

## Efectividad y eficiencia

Efectividad, que cumpla con lo que tiene que hacer.

Eficiencia, hágalo de la mejor manera.

Se pueden estudiar los algoritmos en términos de eficiencia o rendimiento.

El compilador, el OS, el tamaño del input son algunos de los factores que pueden afectar el rendimiento de un algoritmo.

# Complejidad

## Tiempo (CPU)

Es la evaluación más común.

Se intenta determinar cuánto tiempo tarda el algoritmo en completarse. Se cuenta en términos de ciclo de reloj. Cuantas instrucciones elementales ejecuta este algoritmo

(cuantos ciclos se necesitan para completar, cuantas instrucciones elementales por ciclo)

Hay instrucciones (operaciones elementales) que en el cual el CPU tarta T. T depende de la arquitectura.

## Operaciones elementales

Suma,resta…

Operaciones lógicas (AND, OR …)

Operaciones de corrimiento

Asignaciones de variables

Accesos a estructuras indexadas

Comparaciones 0 > < <= …

Todas estas son realizadas en un tiempo T.

## Espacio (RAM, disco duro)

Es otra evaluación.

# Análisis de algoritmos

Formas estándar:

## Empírico (Bench mark)

Comparo dos algoritmos en un mismo ambiente.

El más simple, pero el que aporta menos.

## Simulacional

Definir casos de prueba. Y a partir de estos casos se tabula y se hace un análisis para estos algoritmos. Ver cómo se comporta según la data que le entrego.

Es más exacto que el empírico, pero no puedo definir casos de prueba para todo el flujo del sistema.

## Analítico

Un modelo matemático que engloba todos los posibles casos para el sistema.

# Algoritmos de ordenamiento

Mantener una estructura ordenada permite una mayor facilidad de búsqueda.

## Condicones que evaluar antes de plantear un algoritmo

### Condición

Bajo que condición voy a comparar los dos elementos. Bajo qué criterio. (tamaño, edad, valor numerico, posición, alfabéticamente, etc)

### Tamaño de datos por ordenar

Que tantos datos y que tan grandes son los que voy a ordernar.

# Big O

Notación asintótica, es el grado de complejidad de un algoritmo.

Toma el n más complejo.

Para comparar algoritmos de ordenamiento se puede hacer en términos de cuantas comparaciones hago, pero también puede ser por T.

## Selection Sort

Como uno ordenaría una lista de gente. Pero hecho desde una misma lista.

Buscar el menor y comenzar del siguiente índice.

Comparaciones: n(n-1) /2

Complejidad: O = n^2

## Bubble sort

Tienen la misma complejidad de la selección sort O =n^2.

Hace la misma cantidad de comparaciones, que el selection

Hace swap inmediatos

Si no hago swaps en una iteración puedo levantar un “flag” para detener el algoritmo (no es estándar)

## Insertion Sort

Es más eficiente con estructuras preordenadas.

Su complejidad también es O n^2

(estos últimos 3 tienen una parte ordenada)

### Ejercicio comparativo

18-11-5-27-49-34-8-32-3-29

Selection

S 8

C 45

Bubble

S 22

C 45

Insertion

S 22

C 28

## Shell sort

Donald L. Shell. Basado con el insertion sort

Bueno para arreglos de tamaño medio.

Toma un h = n/2 (esta es solo una forma de elegir h) y comparo el primero con la posición h

Luego muevo la línea y comparo en distancia h.

Cuando termino (llego al final) uso una nueva distancia h = h/2

Cuando h es 1 si hago swap comparo hacia atrás (guardando de donde hice swap).

No tiene una complejidad estándar, ya que el h puede ser calculado de varias formas.

## Merge Sort

Divido la lista a la mitad y aplico el merge sort (es decir vuelvo a dividir) hasta llegar a descomponerlo en unidades básicas (un solo número). Y comparo con la siguiente pareja. Y uno, y sigo comparando en parejas hasta orden todo.

O = nLog2n

## Radix Sort

It has a diffrent approach

No se trabajan los numero como un solo número, sino que trabaja por unidades, decenas, centenas…. Descompone los números.

10 es el radix del sistema decimal (base)

2 es el radix del sistema binario (base)

Crea un arreglo de 10 posiciones, uno para cada digito posible en la unidad.

Y se cuentan cuantos elementos terminan en esa unidad.

Suma secuencialmente.

Toma el último número, y lo acomoda por su digito final en la posición indicada al restarle 1

Ejemplo:

PLANTILLA

[lista]

[0,9]

[0,7]

UNIDADES

170 45 75 90 802 24 2 66 Arreglo original

2 0 2 0 1 2 1 0 0 0 cantidades de números que terminan con esa unidad

2 2 4 4 5 7 8 8 8 8 Lista de la primera iteración: (suma las cantidades de atrás para adelante) (ultimo numero tiene el tamaño del array)

170 90 802 2 24 45 75 66 Fin de la primera iteración

DECENAS

170 90 802 2 24 45 75 66

2 0 1 0 1 0 1 2 0 1 (por decena)

2 3 3 4 4 5 7 7 8

802 2 24 45 66 170 75 90

CENETENAS

802 2 24 45 66 170 75 90

6 1 0 0 0 0 0 0 0 0 (por centena)

6 7 7 7 7 7 7 7 8 8

2 24 45 66 75 90 170 802

Misma complejidad que el Quicksort O = nlog2n

Pero consume el doble de memoria que el Quicksort

## Quick Sort

Es el más popular.

Es parte el array en dos y se llama recursivamente en cada parte.

Se numeran los índices dela

Pivote: ind min + ((ind max-ind min)/2), esta será la posición del pivote

Hay un apuntador al índice más pequeño

Ejemplo:

14 7 3 12 9 11 6 2

Misma complejidad que el Merge Sort y Radix Sort

O = nlog2n

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Descripción | Complejidad | Extra | Código | Gif |
| Bubble Sort | El más grande para atrás., con swaps inmediatos. | N^2 | Va formando una sección ordenada | <https://www.javatpoint.com/bubble-sort-in-java> | <https://external-preview.redd.it/u2fOuJHSWLOiXpDKhx61_nWctTI9-OA9PeVgW0xUll8.gif?format=mp4&s=98fc8939335ffd639ae00faa12734c627ababdd8> |
| Selection Sort | Busca el más pequeño de todos y hace un único swap con el que tenga el apuntador. | N^2 | Va formando una sección ordenada |  |  |
| Insertion Sort | Asume el primero ordenado y compara “hacia atrás” haciendo swaps hasta que el numero que insertado en la parte ordenada. | N^2 | Va formando una sección ordenada |  |  |
| Shell Sort | Hace swaps en una a través de una distancia h que se va decremento hasta llegar a 1, y cuando es 1 (si hace swap) hace un pequeño insertion sort y continua. | Depende del método de como se escoge el h | h = n/2  Bueno para arreglos de tamaño medio. |  |  |
| Merge Sort | Divido la lista a la mitad y mitad de esa mitad hasta tener unidades y después comparo y fusiono hasta tener toda la lista ordenada. | nLog2n | - |  |  |
| Radix Sort | Trabaja por bases.  Ordena por unidades, pues decenas… so on. Ud sabe. | nLog2n | Consume el doble de memoria que el Quicksort |  |  |
| Quick Sort |  | nLog2n | Es el más popular |  |  |

<https://giphy.com/gifs/algorithm-ezjd4NlY4w3io>

# Algoritmos de búsqueda

## Binary search

Asume que se esta trabajando sobre una estructura ordenada.

Busca en la mitad del arreglo, y compara el valor de la mitad con el input

Y descarta la mitad donde el input no se encuentra, y trabaja sobre el sub array izq o derecho según el middle

O(log(N))

## Búsqueda por interpolación

Intenta buscar un mejor pívot. Busca quedar más cerca de lo que se busca.

Es una mejora al binary.

Formula

Middle = low + ((number – array[low]) \* (high - low))/array[high] – array[low]);

O(n)

Ejemplo:

Searching for 28

5 6 9 11 15 18 20 25 28 39

## Hash Search

Es una estructura, que tiene la función de mapear un conjunto grande de números a un conjunto más pequeño.

Convierte llaves en índices. Y varias llaves pueden ser asociados a un mismo índice. Y cada espacio que lleva una llave se llama Bucket.

No se usa función identidad.

### Funciones de hash

Restas sucesivas (identidad)

Aritmética modular (índice es el módulo)

El numero primo determina el tamaño del bucket.

### Mid square method

Eleve al cuadrado la llave tome los r dígitos del centro

Los posibles bucktes🡪 (2^ R -1) buckets

### Trucation method

Ignore parte de la llave

### Folding method

Divide la llave en partes y combine esas partes con operaciones aritméticas

Descarta el sobrante

Problema del hash, colisiones, dos llaves asociadas al mismo índice

Pero son inevitables. Hay que gestionarlas