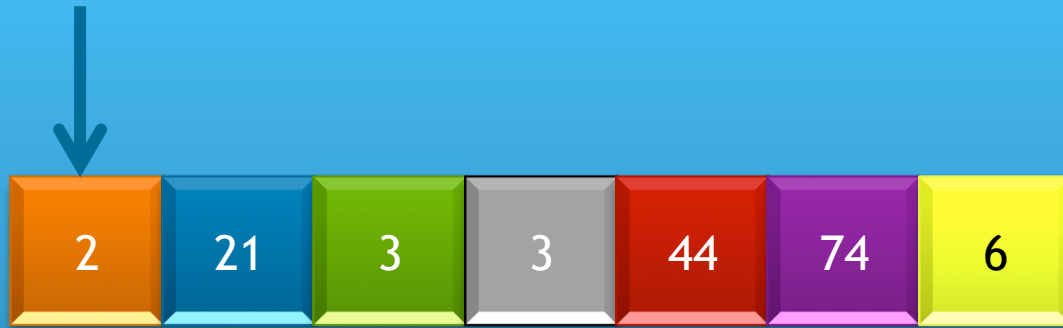




Estructuras de Datos & Complejidad Computacional

Profesor: Patricio Galeas

CAPÍTULO 2 : Arreglos



¿Que son los Arreglos?

- Son las estructuras de datos **más usadas**.
- Es un buen elemento de partida para **entender las ED** en P00.

Ejemplo : Entrenador de futbol, que quiere (en su notebook) mantener el estado de los jugadores que vienen a practicar.

- **Insertar** un jugador cuando llegue a entrenar.
- Verificar (**buscar**) si un jugador se encuentra en el entrenamiento.
- **Eliminar** al jugador cuando sale del campo de entrenamiento.



¿Que son los Arreglos?

- Veamos un ejemplo concreto, con las tres operaciones básicas: **insertar**, **buscar**, **eliminar** ...

AppletViewer: Array.class

New Fill Ins Find Del ☐ Dups OK ☒ No dups Number:

Press any button

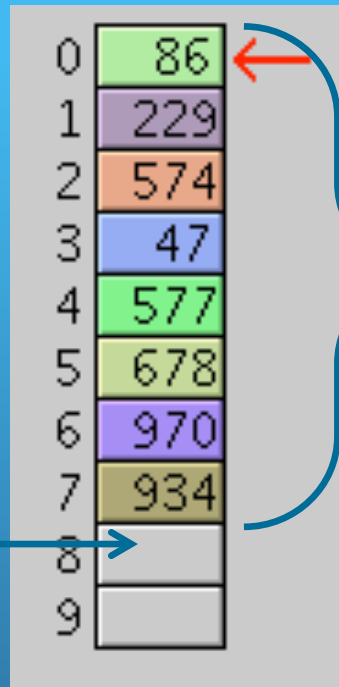
0	994	12	<input type="text"/>
1	505	13	<input type="text"/>
2	477	14	<input type="text"/>
3	860	15	<input type="text"/>
4	112	16	<input type="text"/>
5	857	17	<input type="text"/>
6	802	18	<input type="text"/>
7	485	19	<input type="text"/>
8	219		
9	910		
10			
11			

Subprograma iniciado.



Arreglos : tiempo de las operaciones

Insertar : es rápido
(siempre al final)



0	86
1	229
2	574
3	47
4	577
5	678
6	970
7	934
8	
9	

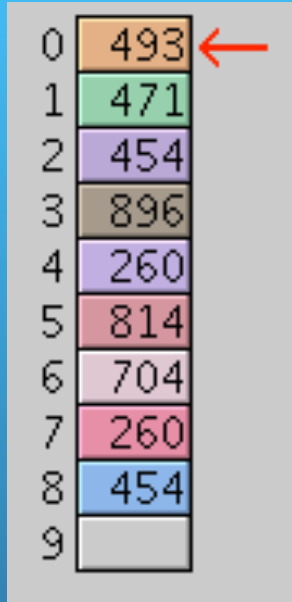
Buscar : necesitamos
en el peor de los casos N
comparaciones, en caso
promedio $N/2$

Borrar: requiere buscar ($N/2$) el
elemento a borrar y eliminar el
espacio vacío, trasladando ($N/2$)
los ítems situados después de la
eliminación.

¿qué pasa cuando es posible ingresar elementos duplicados?

Arreglos : tiempo de las operaciones

Con duplicados



0	493
1	471
2	454
3	896
4	260
5	814
6	704
7	260
8	454
9	

- La **búsqueda se complica**, hay que recorrer siempre todo el arreglo (N).
- La **inserción funciona igual** que en el caso de sin duplicados.
- La **eliminación se complica** en el caso de borrar un elemento duplicado. Hay que hacer desplazamientos por cada eliminación.

	Sin duplicados	Con duplicados
Búsqueda	N/2 comparaciones	N comparaciones
Inserción	Sin comparaciones, un movimiento	Sin comparaciones, un movimiento
Eliminación	N/2 comparaciones, N/2 movimientos	N comparaciones, más de N/2 movimientos

Arreglos : lo básico en Java

Crear un arreglo

```
int[] intArray = new int[100];
```

Obtener el tamaño de un arreglo

```
int arrayLength = intArray.length;    // find array size
```

Acceder a los elementos del arreglo

```
temp = intArray[3];    // get contents of fourth element of array  
intArray[7] = 66;      // insert 66 into the eighth cell
```

Inicializar el arreglo

```
int[] intArray = { 0, 3, 6, 9, 12, 15, 18, 21, 24, 27 };
```

Arreglos : algunos ejemplos



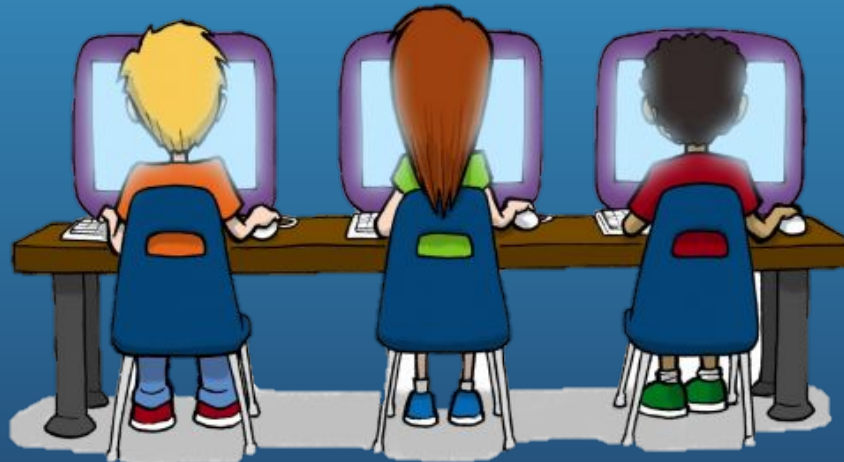
libros/Lafore/ReaderFiles/Chap02/Array/**array.java**



libros/Lafore/ReaderFiles/Chap02/Array/**lowArray.java**

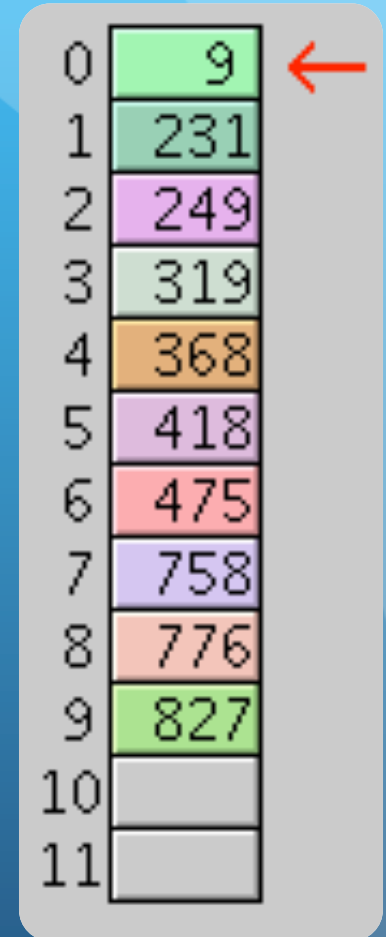


libros/Lafore/ReaderFiles/Chap02/Array/**highArray.java**



Arreglos : ordenados

- Sus elementos están **ordenados ascendentemente**.
- Para **insertar**, hay que **encontrar la posición adecuada**. Todos los **elementos mayores deben ser movidos** para hacer espacio.
- Pero, ¿para que ordenar?
 - Porque **Incrementa** considerablemente la **velocidad de búsqueda** (búsqueda binaria)

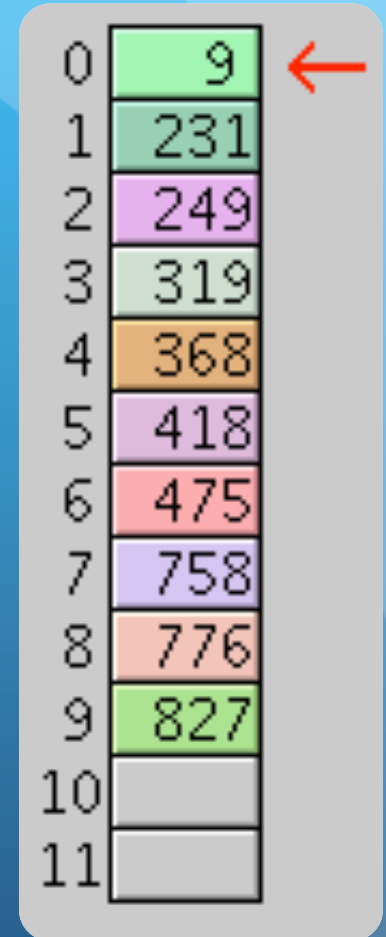


0	9
1	231
2	249
3	319
4	368
5	418
6	475
7	758
8	776
9	827
10	
11	

Arreglos : ordenados -> búsqueda

- **Búsqueda Linear**

- Similar a la del arreglo no ordenado.
- **Pero, “termina”**, si un elemento mayor al buscado es encontrado.
- Insertar **requiere mover los elementos** posteriores.
- Eliminar funciona **similar al arreglo no ordenado**. Sólo **es más rápido** en buscar el elemento a borrar.



0	9
1	231
2	249
3	319
4	368
5	418
6	475
7	758
8	776
9	827
10	
11	

Arreglos : ordenados -> búsqueda

• Búsqueda Binaria

- La **ventaja de ordenar un arreglo** se hace efectiva al utilizar la búsqueda binaria.
- Se basa en el **juego de adivinar un** numero, dividiendo el espacio de alternativas a la mitad en cada iteración.

Buscando el elemento “7”



Paso 1 :	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td></tr></table>	1	2	3	4	5	6	7	8	9	10	11	12	Es 6 : es mayor
1	2	3	4	5	6	7	8	9	10	11	12			
Paso 2 :	<table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td></tr></table>							7	8	9	10	11	12	Es 9 : es menor
						7	8	9	10	11	12			
Paso 3 :	<table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td>7</td><td>8</td><td>9</td><td></td><td></td><td></td></tr></table>							7	8	9				Es 8 : es menor
						7	8	9						
Paso 4 :	<table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td>7</td><td></td><td></td><td></td><td></td><td></td></tr></table>							7						Es 7 : correcto!
						7								

Arreglos : ordenados -> búsqueda

- **Búsqueda Binaria**

- Adivinando un número **33** en un arreglo de 100 elementos

Paso	Numero adivinado	Resultado	Rango posible de valores
0			1 - 100
1	50	muy alto	1 - 49
2	25	muy bajo	26 - 49
3	37	muy alto	26 - 36
4	31	muy bajo	32 - 36
5	34	muy alto	32 - 33
6	32	muy bajo	33 - 33
7	33	correcto!	

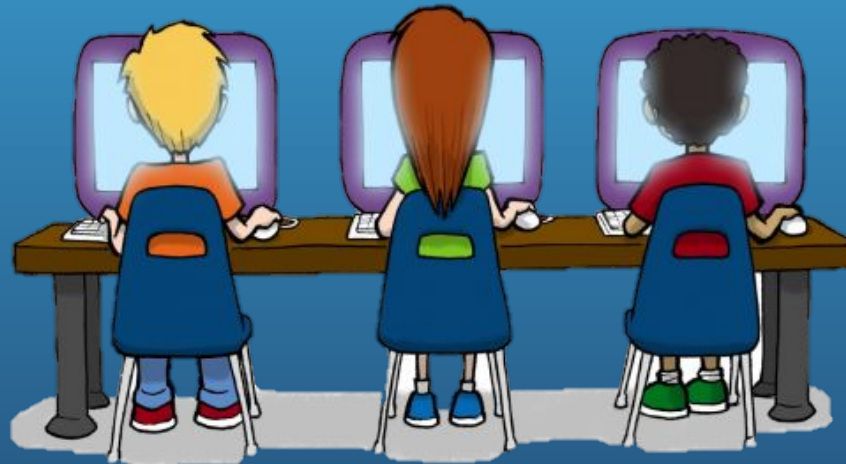
Arreglos : ordenados -> búsqueda

- Búsqueda Binaria

- Ejemplo de código



libros/Lafore/ReaderFiles/Chap02/OrderedArray/**orderedArray.java**



Arreglos : ordenados -> pro/contra



- La **búsqueda** es mucho más **rápida**.
- La **inserción** es más **lenta**. Hay que desplazar a los elementos mayores
- La **eliminación** es en ambos arreglos **lenta**. Hay que desplazar elementos para ocupar el espacio dejado.

Arreglos ordenados son útiles en situaciones de búsquedas frecuentes y donde las inserciones y eliminaciones son poco frecuentes.

Ejemplos:

- **Buen candidato:** Sistema de manejo de empleados de una compañía.
- **Mal candidato:** Sistema de manejo de productos de una tienda de retail.

Arreglos : logaritmos y tiempo de búsqueda

- Hemos visto que la **búsqueda binaria es más rápida** que la secuencial.
- De **1 a 100** : máximo **de 7 pasos** para adivinar un número.
¿Que pasa con otros rangos?

Rango	Comparaciones Necesarias
10	4
100	7
1.000	10
10.000	14
100.000	17
1.000.000	20
10.000.000	24
100.000.000	27
1.000.000.000	30

**¿ Como
calculamos
comparaciones
intermedias ?**

Arreglos : logaritmos y tiempo de búsqueda

Paso s , el mismo que $\log_2(r)$	Rango r	Rango expresado como potencia de 2 (2^s)
0	1	2^0
1	2	2^1
2	4	2^2
3	8	2^3
4	16	2^4
5	32	2^5
6	64	2^6
7	128	2^7
8	256	2^8
9	512	2^9
10	1024	2^{10}

Rango en que puedo adivinar/
encontrar un numero en una
cantidad de s pasos:

$$r = 2^s$$

Número de pasos para
adivinar/encontrar un numero
en un rango r

$$s = \log_2(r)$$



$$3.322 \log_{10}(x) = \log_2(x)$$

(redondeado
al entero
mayor)

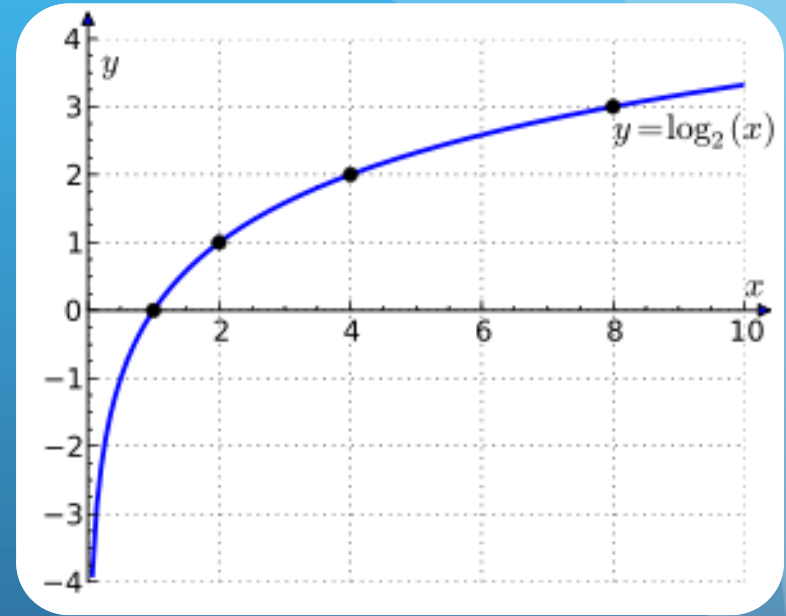
Arreglos : logaritmos y tiempo de búsqueda

- Volvamos a la primera tabla ...

Rango	Comparaciones Necesarias
10	4
100	7
1.000	10
10.000	14
100.000	17
1.000.000	20
10.000.000	24
100.000.000	27
1.000.000.000	30

crece
x10

crece
+3, +4



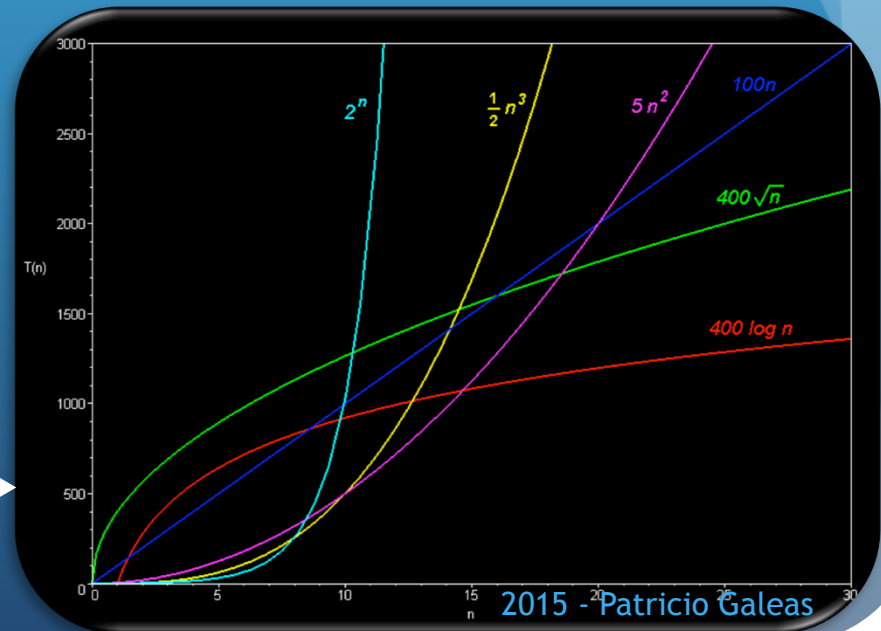
La notación de **Landau**, Big O (O grande) o **cota superior asintótica** se usa normalmente para comparar algoritmos.

$$O(g(x)) = \left\{ f(x) : \text{existen } c, x_0 > 0 \text{ tales que} \right. \\ \left. \forall x \geq x_0 : 0 \leq |f(x)| \leq c|g(x)| \right\}$$

Arreglos

Notación BIG O

- Así como los **automóviles** son categorizados por **tamaños** (citycar, compactos, 4x4, deportivos, etc.)
- La **eficiencia** de los **algoritmos** puede ser medida a través de la notación **Big O**.
- ¿ Por que no se usa una comparación más fácil como:
A es el doble mas eficiente
que **B** ?
- El problema es que la eficiencia **depende del numero de items**.



Arreglos

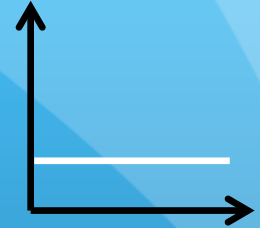
Notación BIG O

- EJEMPLOS

- **Inserción en arreglo no ordenado** : Constante

No depende del # de items.

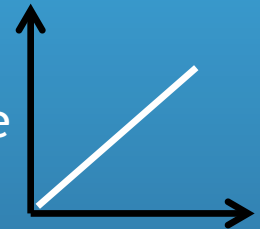
$T = K$, donde K depende del procesador.



- **Búsqueda Lineal** : Proporcional a N

El # de comparaciones es en promedio la mitad del # de items (N).

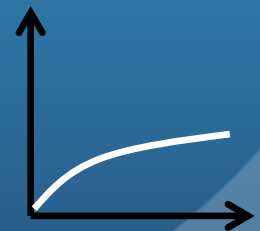
$T = K * N/2$, simplificando $T = K * N$



- **Búsqueda Binaria** : Proporcional a $\log(N)$

$T = K * \log_2(N)$, pero como la transformación de un logaritmo a otro se hace a través de la constante 3.322

$T = K * \log(N)$



Arreglos

Notación BIG O

La constante K no es importante

- La notación Big O no considera la constante K, ya que para comparar algoritmos, el micropocesador o compilador no es relevante.
- Finalmente la letra O mayúscula utilizada por esta notación se puede traducir como : “orden de”

Algoritmo	Tiempo de ejecución en la notación Big O
Búsqueda Linear	$O(N)$
Búsqueda Binaria	$O(\log N)$
Inserción en arreglo no ordenado	$O(1)$
Inserción en arreglo ordenado	$O(N)$
Eliminación en arreglo no ordenado	$O(N)$
Eliminación en arreglo ordenado	$O(N)$

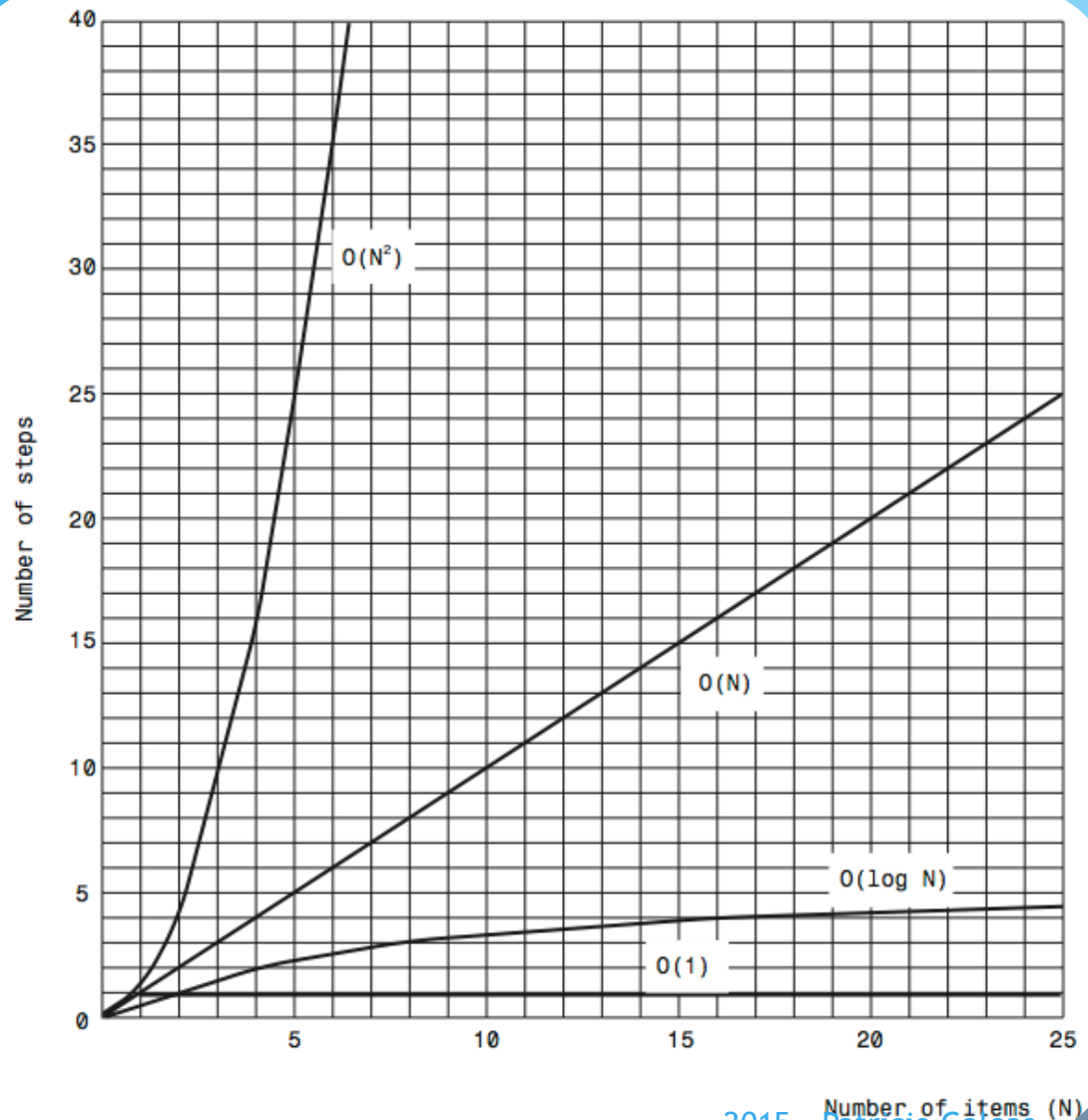
Arreglos

Notación BIG O

Uno podría decir:

- $O(1)$ es excelente
- $O(\log N)$ es bueno
- $O(N)$ es regular
- $O(N^2)$ es malo

La idea de usar Big O no es comparar tiempos de ejecución sino para transmitir cómo el tiempo de ejecución se ve afectado por el número de items.



Arreglos : Resumen



- Los **arreglos en Java** son objetos.
- Los **arreglos no ordenados** ofrecen inserción rápida pero búsqueda y eliminación lenta.
- La **búsqueda binaria** puede ser aplicada a arreglos ordenados.
- El **logaritmo en base B de un numero A** es (aproximadamente) el numero de veces que uno puede dividir el numero A por B, antes de que el resultado sea menor a 1.
- La **búsqueda lineal** requiere un tiempo proporcional número de ítems
- La **búsqueda binaria** requiere un tiempo proporcional al logaritmo del número de ítems.
- La **notación Big O**, es una buena forma de comparar la velocidad de algoritmos.
- Un **algoritmo que de orden**: $O(1)$ es excelente, $O(\log N)$ es bueno, $O(N)$ es regular y $O(N^2)$ es malo.

Arreglos : Experimentos

- Use el **Array Workshop applet** para insertar, buscar y eliminar items. Trate de predecir que va a ocurrir en cada paso. Hacer lo mismo para arreglos que con o sin duplicados.
- Use el **Ordered Workshop applet** y asegúrese de poder predecir el rango que se seleccionará en cada paso.
- Usando el **Ordered Workshop applet** y tomado un numero impar de elementos (no existe un elemento intermedio), investigue qué elemento examinará primero la búsqueda binaria.
- **Revisar** la implementación de la **clase Persona** en **classDataArray.java**

