

Programación de Computadores: Java - Búsqueda y Ordenamiento

Juan F. Pérez

Departamento MACC
Matemáticas Aplicadas y Ciencias de la Computación
Universidad del Rosario

juanferna.perez@urosario.edu.co

Segundo Semestre de 2017

Contenidos

1 Búsqueda

2 Ordenamiento

- Ordenamiento por selección
- Ordenamiento por Inserción

Búsqueda

Búsqueda

Dado un arreglo de enteros

Búsqueda

Dado un arreglo de enteros

Escribir un método que busque un entero llave

Búsqueda

Dado un arreglo de enteros

Escribir un método que busque un entero llave

El resultado es el índice de la primera posición del arreglo donde se encuentre la llave

Búsqueda

Dado un arreglo de enteros

Escribir un método que busque un entero llave

El resultado es el índice de la primera posición del arreglo donde se encuentre la llave

Si el la llave no está presente en el arreglo, el método retorna -1

Búsqueda

Dado un arreglo de enteros

Escribir un método que busque un entero llave

El resultado es el índice de la primera posición del arreglo donde se encuentre la llave

Si el la llave no está presente en el arreglo, el método retorna -1

Ejemplo:

Búsqueda

Dado un arreglo de enteros

Escribir un método que busque un entero llave

El resultado es el índice de la primera posición del arreglo donde se encuentre la llave

Si el la llave no está presente en el arreglo, el método retorna -1

Ejemplo:

Arreglo: [3, 5, 1, 6, 3, 7]

Búsqueda

Dado un arreglo de enteros

Escribir un método que busque un entero llave

El resultado es el índice de la primera posición del arreglo donde se encuentre la llave

Si el la llave no está presente en el arreglo, el método retorna -1

Ejemplo:

Arreglo: [3, 5, 1, 6, 3, 7]

Llave: 6, retorna 3

Búsqueda

Dado un arreglo de enteros

Escribir un método que busque un entero llave

El resultado es el índice de la primera posición del arreglo donde se encuentre la llave

Si el la llave no está presente en el arreglo, el método retorna -1

Ejemplo:

Arreglo: [3, 5, 1, 6, 3, 7]

Llave: 6, retorna 3

Llave: 7, retorna 5

Búsqueda

Dado un arreglo de enteros

Escribir un método que busque un entero llave

El resultado es el índice de la primera posición del arreglo donde se encuentre la llave

Si el la llave no está presente en el arreglo, el método retorna -1

Ejemplo:

Arreglo: [3, 5, 1, 6, 3, 7]

Llave: 6, retorna 3

Llave: 7, retorna 5

Llave: 8, retorna -1

Búsqueda

Más general: no solo enteros, aplicable a cualquier objeto

Búsqueda

Más general: no solo enteros, aplicable a cualquier objeto
Necesitamos una forma de saber si dos objetos son iguales

Búsqueda

Más general: no solo enteros, aplicable a cualquier objeto
Necesitamos una forma de saber si dos objetos son iguales
Método `equals()`

Búsqueda

Más general: no solo enteros, aplicable a cualquier objeto

Necesitamos una forma de saber si dos objetos son iguales

Método `equals()`

Clase `Object()`

Búsqueda

Más general: no solo enteros, aplicable a cualquier objeto

Necesitamos una forma de saber si dos objetos son iguales

Método `equals()`

Clase `Object()`

<https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html>

Cómo buscar un elemento?

Recibimos un arreglo $A = (a_1, \dots, a_n)$ y una llave b

Cómo buscar un elemento?

Recibimos un arreglo $A = (a_1, \dots, a_n)$ y una llave b

Revisar uno por uno de los elementos a_i , empezando con a_1

Cómo buscar un elemento?

Recibimos un arreglo $A = (a_1, \dots, a_n)$ y una llave b

Revisar uno por uno de los elementos a_i , empezando con a_1

Si $a_i = b$, retornamos i

Cómo buscar un elemento?

Recibimos un arreglo $A = (a_1, \dots, a_n)$ y una llave b

Revisar uno por uno de los elementos a_i , empezando con a_1

Si $a_i = b$, retornamos i

Si terminamos de revisar el arreglo sin encontrar b , retornamos -1

Cómo buscar un elemento?

Recibimos un arreglo $A = (a_1, \dots, a_n)$ y una llave b

Revisar uno por uno de los elementos a_i , empezando con a_1

Si $a_i = b$, retornamos i

Si terminamos de revisar el arreglo sin encontrar b , retornamos -1

Búsqueda Lineal

Búsqueda Lineal

Pseudo-código:

Versión completa del algoritmo

Búsqueda Lineal

Pseudo-código:

- Versión completa del algoritmo

- Insumo (input) y resultado (output)

Búsqueda Lineal

Pseudo-código:

- Versión completa del algoritmo

- Insumo (input) y resultado (output)

- Similar a código pero con sintaxis más simple

Búsqueda Lineal

Pseudo-código:

Versión completa del algoritmo

Insumo (input) y resultado (output)

Similar a código pero con sintaxis más simple

```
function BUSQUEDALINEAL( $A = (a_1, \dots, a_n)$ ,  $b$ )  
  for  $i = 1 \dots n$  do  
    if  $a_i = b$  then  
      return  $i$   
    end if  
  end for  
  return  $-1$   
end function
```

Búsqueda Lineal - Implementación

Implementación: EjemploBusquedaLineal

```
private int busquedaLineal(int llave, int[] arreglo) {  
    for(int i = 0; i < arreglo.length; i++){  
        if(llave == arreglo[i]) return i;  
    }  
    return -1;  
}
```

Búsqueda Lineal - Implementación

Implementación: EjemploBusquedaLinealV2

```
private int busquedaLineal(String llave, String[] arreglo) {  
    for(int i = 0; i < arreglo.length; i++){  
        if(llave.equals(arreglo[i])) return i;  
    }  
    return -1;  
}
```

Búsqueda Lineal - Análisis

```
function BUSQUEDALINEAL( $A = (a_1, \dots, a_n)$ ,  $b$ )  
  for  $i = 1 \dots n$  do  
    if  $a_i = b$  then  
      return  $i$   
    end if  
  end for  
  return  $-1$   
end function
```

Operaciones:

$i = 1$ (1 vez)

Búsqueda Lineal - Análisis

```
function BUSQUEDALINEAL( $A = (a_1, \dots, a_n)$ ,  $b$ )  
  for  $i = 1 \dots n$  do  
    if  $a_i = b$  then  
      return  $i$   
    end if  
  end for  
  return  $-1$   
end function
```

Operaciones:

$i = 1$ (1 vez)

Comparación: $a_i = b$ (en el peor caso n veces)

Búsqueda Lineal - Análisis

```
function BUSQUEDALINEAL( $A = (a_1, \dots, a_n)$ ,  $b$ )  
  for  $i = 1 \dots n$  do  
    if  $a_i = b$  then  
      return  $i$   
    end if  
  end for  
  return  $-1$   
end function
```

Operaciones:

$i = 1$ (1 vez)

Comparación: $a_i = b$ (en el peor caso n veces)

$i = i + 1$ (en el peor caso n veces)

Búsqueda Lineal - Análisis

```
function BUSQUEDALINEAL( $A = (a_1, \dots, a_n)$ ,  $b$ )  
  for  $i = 1 \dots n$  do  
    if  $a_i = b$  then  
      return  $i$   
    end if  
  end for  
  return  $-1$   
end function
```

Operaciones:

$i = 1$ (1 vez)

Comparación: $a_i = b$ (en el peor caso n veces)

$i = i + 1$ (en el peor caso n veces)

return (1 vez)

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

$$i = 1 \text{ (1 vez - costo } c_1)$$

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

$i = 1$ (1 vez - costo c_1)

Comparación: $a_i = b$ (n veces - costo c_2)

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

$$i = 1 \text{ (1 vez - costo } c_1)$$

$$\text{Comparación: } a_i = b \text{ (} n \text{ veces - costo } c_2)$$

$$i = i + 1 \text{ (} n \text{ veces - costo } c_3)$$

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

$i = 1$ (1 vez - costo c_1)

Comparación: $a_i = b$ (n veces - costo c_2)

$i = i + 1$ (n veces - costo c_3)

return (1 vez - costo c_4)

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

$i = 1$ (1 vez - costo c_1)

Comparación: $a_i = b$ (n veces - costo c_2)

$i = i + 1$ (n veces - costo c_3)

return (1 vez - costo c_4)

Costo total:

$$c_1 + c_4 + n(c_3 + c_4)$$

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

$i = 1$ (1 vez - costo c_1)

Comparación: $a_i = b$ (n veces - costo c_2)

$i = i + 1$ (n veces - costo c_3)

return (1 vez - costo c_4)

Costo total:

$$c_1 + c_4 + n(c_3 + c_4)$$

Si n es grande, constantes dejan de ser importantes:

$$n(c_3 + c_4)$$

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

$i = 1$ (1 vez - costo c_1)

Comparación: $a_i = b$ (n veces - costo c_2)

$i = i + 1$ (n veces - costo c_3)

return (1 vez - costo c_4)

Costo total:

$$c_1 + c_4 + n(c_3 + c_4)$$

Si n es grande, constantes dejan de ser importantes:

$$n(c_3 + c_4)$$

Solo nos interesa el orden:

$$n$$

Búsqueda Lineal - Análisis - Notación O

Costo total:

Solo nos interesa el orden:

n

Búsqueda Lineal - Análisis - Notación O

Costo total:

Solo nos interesa el orden:

$$n$$

Notación O :

$$O(n)$$

Búsqueda Lineal - Análisis - Notación O

Costo total:

Solo nos interesa el orden:

$$n$$

Notación O :

$$O(n)$$

El costo (número de operaciones - tiempo) de buscar un ítem en una lista de tamaño n crece linealmente con n

Búsqueda Lineal - Análisis - Notación O

Costo total:

Solo nos interesa el orden:

n

Notación O :

$O(n)$

El costo (número de operaciones - tiempo) de buscar un ítem en una lista de tamaño n crece linealmente con n

Tamaño (n)	Costo
10	10
1.000	1.000
100.000	100.000
100.000.000	100.000.000

¿Método alternativo de Búsqueda?

¿Es posible mejorar la búsqueda?

¿Método alternativo de Búsqueda?

¿Es posible mejorar la búsqueda?

Revisemos nuevamente `EjemploBusquedaLinealV2`

¿Método alternativo de Búsqueda?

¿Es posible mejorar la búsqueda?

Revisemos nuevamente EjemploBusquedaLinealV2

```
{ "Armenia", "Barranquilla",  
  "Bogota", "Bucaramanga", "Buenaventura", "Cali",  
  "Cartagena", "Cucuta", "Florenzia", "Ibague"  
}
```

¿Método alternativo de Búsqueda?

¿Es posible mejorar la búsqueda?

Revisemos nuevamente EjemploBusquedaLinealV2

```
{ "Armenia", "Barranquilla",  
  "Bogota", "Bucaramanga", "Buenaventura", "Cali",  
  "Cartagena", "Cucuta", "Florenzia", "Ibague"  
}
```

¿Tiene la lista una característica que podamos usar?

¿Método alternativo de Búsqueda?

¿Es posible mejorar la búsqueda?

Revisemos nuevamente EjemploBusquedaLinealV2

```
{ "Armenia", "Barranquilla",  
  "Bogota", "Bucaramanga", "Buenaventura", "Cali",  
  "Cartagena", "Cucuta", "Florenzia", "Ibague"  
}
```

¿Tiene la lista una característica que podamos usar?

Ordenada

¿Método alternativo de Búsqueda?

```
{"Armenia", "Barranquilla", "Bogota", "Bucaramanga",  
"Buenaventura", "Cali", "Cartagena", "Cucuta",  
"Florenceia", "Ibague"}
```

Busquemos *Florenceia*

¿Método alternativo de Búsqueda?

```
{"Armenia", "Barranquilla", "Bogota", "Bucaramanga",  
"Buenaventura", "Cali", "Cartagena", "Cucuta",  
"Florencia", "Ibague"}
```

Busquemos *Florencia*

La posición media es $\frac{0+9}{2} = 4$ (entero)

¿Método alternativo de Búsqueda?

```
{"Armenia", "Barranquilla", "Bogota", "Bucaramanga",  
"Buenaventura", "Cali", "Cartagena", "Cucuta",  
"Florenia", "Ibague"}
```

Busquemos *Florenia*

La posición media es $\frac{0+9}{2} = 4$ (entero)

Elemento en la posición 4: Bucaramanga

¿Método alternativo de Búsqueda?

```
{"Armenia", "Barranquilla", "Bogota", "Bucaramanga",  
"Buenaventura", "Cali", "Cartagena", "Cucuta",  
"Florencia", "Ibague"}
```

Busquemos *Florencia*

La posición media es $\frac{0+9}{2} = 4$ (entero)

Elemento en la posición 4: Bucaramanga

¿Está Florencia antes o después de Bucaramanga?

¿Método alternativo de Búsqueda?

```
{"Armenia", "Barranquilla", "Bogota", "Bucaramanga",  
"Buenaventura", "Cali", "Cartagena", "Cucuta",  
"Florencia", "Ibague"}
```

Busquemos *Florencia*

La posición media es $\frac{0+9}{2} = 4$ (entero)

Elemento en la posición 4: Bucaramanga

¿Está Florencia antes o después de Bucaramanga?

Después

¿Método alternativo de Búsqueda?

```
{ "Armenia", "Barranquilla", "Bogota", "Bucaramanga",  
  "Buenaventura", "Cali", "Cartagena", "Cucuta",  
  "Florencia", "Ibague" }
```

Busquemos *Florencia*

La posición media es $\frac{0+9}{2} = 4$ (entero)

Elemento en la posición 4: Bucaramanga

¿Está Florencia antes o después de Bucaramanga?

Después

Entonces Florencia debe estar en

```
{ "Buenaventura", "Cali", "Cartagena",  
  "Cucuta", "Florencia", "Ibague" }
```

¿Método alternativo de Búsqueda?

```
{ "Buenaventura", "Cali", "Cartagena",  
  "Cucuta", "Florenceia", "Ibague" }
```

Busquemos *Florenceia*

¿Método alternativo de Búsqueda?

```
{ "Buenaventura", "Cali", "Cartagena",  
  "Cucuta", "Florenxia", "Ibague" }
```

Busquemos *Florenxia*

La posición media es $\frac{0+6}{2} = 3$ (entero)

¿Método alternativo de Búsqueda?

```
{ "Buenaventura", "Cali", "Cartagena",  
  "Cucuta", "Florenceia", "Ibague" }
```

Busquemos *Florenceia*

La posición media es $\frac{0+6}{2} = 3$ (entero)

Elemento en la posición 3: Cartagena

¿Método alternativo de Búsqueda?

```
{ "Buenaventura", "Cali", "Cartagena",  
  "Cucuta", "Florenceia", "Ibague" }
```

Busquemos *Florenceia*

La posición media es $\frac{0+6}{2} = 3$ (entero)

Elemento en la posición 3: Cartagena

¿Está Florenceia antes o después de Cartagena?

¿Método alternativo de Búsqueda?

```
{ "Buenaventura", "Cali", "Cartagena",  
  "Cucuta", "Florenceia", "Ibague" }
```

Busquemos *Florenceia*

La posición media es $\frac{0+6}{2} = 3$ (entero)

Elemento en la posición 3: Cartagena

¿Está Florenceia antes o después de Cartagena?

Después

¿Método alternativo de Búsqueda?

```
|| {"Buenaventura", "Cali", "Cartagena",  
   "Cucuta", "Florencia", "Ibague"}
```

Busquemos *Florencia*

La posición media es $\frac{0+6}{2} = 3$ (entero)

Elemento en la posición 3: Cartagena

¿Está Florencia antes o después de Cartagena?

Después

Entonces Florencia debe estar en

```
|| {"Cucuta", "Florencia", "Ibague"}
```

¿Método alternativo de Búsqueda?

```
|| {"Cucuta", "Florencia", "Ibague"}
```

Busquemos *Florencia*

¿Método alternativo de Búsqueda?

```
|| {"Cucuta", "Florenxia", "Ibague"}
```

Busquemos *Florenxia*

La posición media es $\frac{0+3}{2} = 1$ (entero)

¿Método alternativo de Búsqueda?

```
|| {"Cucuta", "Florenxia", "Ibague"}
```

Busquemos *Florenxia*

La posición media es $\frac{0+3}{2} = 1$ (entero)

Elemento en la posición 1: Florenxia

¿Método alternativo de Búsqueda?

```
|| {"Cucuta", "Florencia", "Ibague"}
```

Busquemos *Florencia*

La posición media es $\frac{0+3}{2} = 1$ (entero)

Elemento en la posición 1: Florencia

Terminamos

¿Método alternativo de Búsqueda?

```
|| {"Cucuta", "Florenxia", "Ibague"}
```

Busquemos *Florenxia*

La posición media es $\frac{0+3}{2} = 1$ (entero)

Elemento en la posición 1: Florenxia

Terminamos

Número de comparaciones: 3

¿Método alternativo de Búsqueda?

```
|| {"Cucuta", "Florenxia", "Ibague"}
```

Busquemos *Florenxia*

La posición media es $\frac{0+3}{2} = 1$ (entero)

Elemento en la posición 1: Florenxia

Terminamos

Número de comparaciones: 3

Con búsqueda lineal: 9

Búsqueda Binaria

En cada paso comparamos la llave con el elemento medio de la lista

Búsqueda Binaria

En cada paso comparamos la llave con el elemento medio de la lista
Si elemento medio es el que buscamos, terminamos

Búsqueda Binaria

En cada paso comparamos la llave con el elemento medio de la lista
Si elemento medio es el que buscamos, terminamos
Si la llave es mayor que el elemento medio, descartamos la primera mitad de la lista y mantenemos la segunda mitad

Búsqueda Binaria

En cada paso comparamos la llave con el elemento medio de la lista

Si elemento medio es el que buscamos, terminamos

Si la llave es mayor que el elemento medio, descartamos la primera mitad de la lista y mantenemos la segunda mitad

Si la llave es menor que el elemento medio, descartamos la segunda mitad de la lista y mantenemos la primera mitad

Búsqueda Binaria

En cada paso comparamos la llave con el elemento medio de la lista

Si elemento medio es el que buscamos, terminamos

Si la llave es mayor que el elemento medio, descartamos la primera mitad de la lista y mantenemos la segunda mitad

Si la llave es menor que el elemento medio, descartamos la segunda mitad de la lista y mantenemos la primera mitad

Repetimos con la lista restante

Búsqueda Binaria - Pseudo-código

```
function BUSQUEDABINARIA( $A = (a_1, \dots, a_n)$ ,  $b$ )  
  izq  $\leftarrow$  1  
  der  $\leftarrow$  A.length =  $n$   
  while izq  $\leq$  der do  
    med  $\leftarrow$  (izq+der)/2  
    if  $med = b$  then  
      return med  
    else if  $med < b$  then  
      izq  $\leftarrow$  med+1  
    else  
      der  $\leftarrow$  med-1  
    end if  
  end while  
  return -1  
end function
```


Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

$$\text{izq} \leftarrow 1 \text{ (1 vez - costo } c_1)$$

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

izq \leftarrow 1 (1 vez - costo c_1)

der \leftarrow A.length = n (1 vez - costo c_2)

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

$\text{izq} \leftarrow 1$ (1 vez - costo c_1)

$\text{der} \leftarrow A.\text{length} = n$ (1 vez - costo c_2)

$\text{med} \leftarrow (\text{izq} + \text{der}) / 2$ (k veces - costo c_3)

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

$\text{izq} \leftarrow 1$ (1 vez - costo c_1)

$\text{der} \leftarrow A.\text{length} = n$ (1 vez - costo c_2)

$\text{med} \leftarrow (\text{izq} + \text{der}) / 2$ (k veces - costo c_3)

comparación $\text{med} = b$ (k veces - costo c_4)

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

$\text{izq} \leftarrow 1$ (1 vez - costo c_1)

$\text{der} \leftarrow A.\text{length} = n$ (1 vez - costo c_2)

$\text{med} \leftarrow (\text{izq} + \text{der}) / 2$ (k veces - costo c_3)

comparación $\text{med} = b$ (k veces - costo c_4)

comparación $\text{med} < b$ (k veces - costo c_5)

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

izq \leftarrow 1 (1 vez - costo c_1)

der \leftarrow A.length = n (1 vez - costo c_2)

med \leftarrow (izq+der)/2 (k veces - costo c_3)

comparación med = b (k veces - costo c_4)

comparación med < b (k veces - costo c_5)

return (1 vez - costo c_6)

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

izq \leftarrow 1 (1 vez - costo c_1)

der \leftarrow A.length = n (1 vez - costo c_2)

med \leftarrow (izq+der)/2 (k veces - costo c_3)

comparación med = b (k veces - costo c_4)

comparación med < b (k veces - costo c_5)

return (1 vez - costo c_6)

Costo total:

$$c_1 + c_2 + c_6 + k(c_3 + c_4 + c_5)$$

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

Costo total:

$$c_1 + c_2 + c_6 + k(c_3 + c_4 + c_5)$$

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

Costo total:

$$c_1 + c_2 + c_6 + k(c_3 + c_4 + c_5)$$

Si k es grande, constantes dejan de ser importantes:

$$k(c_3 + c_4 + c_5)$$

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

Costo total:

$$c_1 + c_2 + c_6 + k(c_3 + c_4 + c_5)$$

Si k es grande, constantes dejan de ser importantes:

$$k(c_3 + c_4 + c_5)$$

Solo nos interesa el orden:

$$O(k)$$

Búsqueda Binaria - Análisis

Análisis de peor caso (elemento buscado en la última posición considerada o no está en la lista):

Costo total:

$$c_1 + c_2 + c_6 + k(c_3 + c_4 + c_5)$$

Si k es grande, constantes dejan de ser importantes:

$$k(c_3 + c_4 + c_5)$$

Solo nos interesa el orden:

$$O(k)$$

¿Cuánto vale k en términos de n ?

Búsqueda Binaria - Análisis

¿Cuánto vale k en términos de n ?

Cada iteración dividimos n en 2, k veces

Búsqueda Binaria - Análisis

¿Cuánto vale k en términos de n ?

Cada iteración dividimos n en 2, k veces

$$n/2^k = 1$$

Búsqueda Binaria - Análisis

¿Cuánto vale k en términos de n ?

Cada iteración dividimos n en 2, k veces

$$n/2^k = 1$$

$$n = 2^k$$

Búsqueda Binaria - Análisis

¿Cuánto vale k en términos de n ?

Cada iteración dividimos n en 2, k veces

$$n/2^k = 1$$

$$n = 2^k$$

$$\log_2 n = k$$

Búsqueda Binaria - Análisis

¿Cuánto vale k en términos de n ?

Cada iteración dividimos n en 2, k veces

$$n/2^k = 1$$

$$n = 2^k$$

$$\log_2 n = k$$

$$O(\log_2 n)$$

Búsqueda Binaria - Análisis

¿Cuánto vale k en términos de n ?

Cada iteración dividimos n en 2, k veces

$$n/2^k = 1$$

$$n = 2^k$$

$$\log_2 n = k$$

$$O(\log_2 n)$$

El costo (número de operaciones - tiempo) de buscar un ítem en una lista de tamaño n crece como $\log_2 n$

Búsqueda Binaria - Análisis

El costo (número de operaciones - tiempo) de buscar un ítem en una lista de tamaño n crece como $\log_2 n$

Tamaño (n)	Costo ($\lceil \log_2 n \rceil$)
10	4
1.000	10
100.000	17
100.000.000	27

Búsqueda Binaria - Implementación

EjemploBusquedaBinaria.java

```
private int busquedaBinaria(int llave, int[] arreglo) {
    int izq = 0;
    int der = arreglo.length - 1;
    while(izq <= der){
        int med = (izq+der)/2;
        if (llave == arreglo[med]){
            return med;
        }else if(llave > arreglo[med]){
            izq = med +1;
        }else{
            der = med - 1;
        }
    }
    return -1;
}
```

Búsqueda Binaria - Implementación

En general, necesitamos comparar objetos

Búsqueda Binaria - Implementación

En general, necesitamos comparar objetos
Interfaz Comparable

Búsqueda Binaria - Implementación

En general, necesitamos comparar objetos

Interfaz Comparable

<https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html>

Búsqueda Binaria - Implementación

En general, necesitamos comparar objetos

Interfaz Comparable

<https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html>

Método `compareTo(Object otro)`

Búsqueda Binaria - Implementación

En general, necesitamos comparar objetos

Interfaz Comparable

<https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html>

Método compareTo(Object otro)

Retorna 0 si este objeto es igual a otro

Búsqueda Binaria - Implementación

En general, necesitamos comparar objetos

Interfaz Comparable

<https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html>

Método `compareTo(Object otro)`

Retorna 0 si este objeto es igual a otro

Retorna < 0 si este objeto es menor que otro

Búsqueda Binaria - Implementación

En general, necesitamos comparar objetos

Interfaz Comparable

<https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html>

Método `compareTo(Object otro)`

Retorna 0 si este objeto es igual a otro

Retorna < 0 si este objeto es menor que otro

Retorna > 0 si este objeto es mayor que otro

Búsqueda Binaria - Implementación

EjemploBusquedaBinariaV2.java

```
private int busquedaBinaria(String llave, String[] arreglo)
{
    int izq = 0;
    int der = arreglo.length - 1;
    while(izq <= der){
        int med = (izq+der)/2;
        int cmp = llave.compareTo(arreglo[med]);
        if (cmp == 0){
            return med;
        }else if(cmp > 0){
            izq = med +1;
        }else{
            der = med - 1;
        }
    }
    return -1;
}
```

Búsqueda Lineal vs. Binaria

Binaria (mucho) más eficiente que Lineal

Búsqueda Lineal vs. Binaria

Binaria (mucho) más eficiente que Lineal

Lineal más sencilla de implementar que Binaria

Búsqueda Lineal vs. Binaria

Binaria (mucho) más eficiente que Lineal

Lineal más sencilla de implementar que Binaria

Binaria requiere que el arreglo esté ordenado

Búsqueda Lineal vs. Binaria

Binaria (mucho) más eficiente que Lineal

Lineal más sencilla de implementar que Binaria

Binaria requiere que el arreglo esté ordenado

¿Cómo ordenamos arreglos/listas?

Ordenamiento

Ordenamiento

Consideremos arreglos de enteros

Ordenamiento

Consideremos arreglos de enteros

$$\{15, 67, 82, 27, 90, 48\}$$

Ordenamiento

Consideremos arreglos de enteros

$$\{15, 67, 82, 27, 90, 48\}$$

¿Cómo lo ordenamos?

Ordenamiento

Consideremos arreglos de enteros

$$\{15, 67, 82, 27, 90, 48\}$$

¿Cómo lo ordenamos?

Muchas posibilidades

Ordenamiento

Consideremos arreglos de enteros

$$\{15, 67, 82, 27, 90, 48\}$$

¿Cómo lo ordenamos?

Muchas posibilidades

Una: ordenamiento por selección

Ordenamiento por selección

Dado un arreglo $A = (a_1, \dots, a_n)$

Ordenamiento por selección

Dado un arreglo $A = (a_1, \dots, a_n)$

Seleccionar el elemento menor y ubicarlo en la primera posición, intercambiándolo con a_1

Ordenamiento por selección

Dado un arreglo $A = (a_1, \dots, a_n)$

Seleccionar el elemento menor y ubicarlo en la primera posición, intercambiándolo con a_1

Resultado: primera posición ordenada

Ordenamiento por selección

Dado un arreglo $A = (a_1, \dots, a_n)$

Seleccionar el elemento menor y ubicarlo en la primera posición, intercambiándolo con a_1

Resultado: primera posición ordenada

Repetir con el arreglo $A = (a_2, \dots, a_n)$

Ordenamiento por selección - Pseudo-código

```
function ORDENAMIENTOSELECCION( $A = (a_1, \dots, a_n)$ )  
  for  $i = 1 \dots n$  do  
     $b \leftarrow \text{mín}((a_i, \dots, a_n))$   
     $A \leftarrow \text{intercambiar}(A, a_i, b)$   
  end for  
  return  $A$   
end function
```

Ordenamiento por Selección - Ejemplo

$\{15, 67, 82, 27, 90, 48\}$

Ordenamiento por Selección - Ejemplo

$\{15, 67, 82, 27, 90, 48\}$

$\text{mín}(\{15, 67, 82, 27, 90, 48\}) = 15$

Ordenamiento por Selección - Ejemplo

$\{15, 67, 82, 27, 90, 48\}$

$\text{mín}(\{15, 67, 82, 27, 90, 48\}) = 15$

$\{\mathbf{15}, 67, 82, 27, 90, 48\}$

Ordenamiento por Selección - Ejemplo

$\{15, 67, 82, 27, 90, 48\}$

$\text{mín}(\{15, 67, 82, 27, 90, 48\}) = 15$

$\{\mathbf{15}, 67, 82, 27, 90, 48\}$

$\text{mín}(\{67, 82, 27, 90, 48\}) = 27$

Ordenamiento por Selección - Ejemplo

$\{15, 67, 82, 27, 90, 48\}$

$\text{mín}(\{15, 67, 82, 27, 90, 48\}) = 15$

$\{\mathbf{15}, 67, 82, 27, 90, 48\}$

$\text{mín}(\{67, 82, 27, 90, 48\}) = 27$

$\{15, \mathbf{27}, 82, \mathbf{67}, 90, 48\}$

Ordenamiento por Selección - Ejemplo

$\{15, 67, 82, 27, 90, 48\}$

$\text{mín}(\{15, 67, 82, 27, 90, 48\}) = 15$

$\{\mathbf{15}, 67, 82, 27, 90, 48\}$

$\text{mín}(\{67, 82, 27, 90, 48\}) = 27$

$\{15, \mathbf{27}, 82, \mathbf{67}, 90, 48\}$

$\text{mín}(\{82, 67, 90, 48\}) = 48$

Ordenamiento por Selección - Ejemplo

$\{15, 67, 82, 27, 90, 48\}$

$\text{mín}(\{15, 67, 82, 27, 90, 48\}) = 15$

$\{\mathbf{15}, 67, 82, 27, 90, 48\}$

$\text{mín}(\{67, 82, 27, 90, 48\}) = 27$

$\{15, \mathbf{27}, 82, \mathbf{67}, 90, 48\}$

$\text{mín}(\{82, 67, 90, 48\}) = 48$

$\{15, 27, \mathbf{48}, 67, 90, \mathbf{82}\}$

Ordenamiento por Selección - Ejemplo

$\{15, 27, 48, 67, 90, 82\}$

Ordenamiento por Selección - Ejemplo

$\{15, 27, 48, 67, 90, 82\}$

$\text{mín}(\{67, 90, 82\}) = 67$

Ordenamiento por Selección - Ejemplo

$\{15, 27, 48, 67, 90, 82\}$

$\text{mín}(\{67, 90, 82\}) = 67$

$\{15, 27, 48, \mathbf{67}, 90, 82\}$

Ordenamiento por Selección - Ejemplo

$\{15, 27, 48, 67, 90, 82\}$

$\text{mín}(\{67, 90, 82\}) = 67$

$\{15, 27, 48, \mathbf{67}, 90, 82\}$

$\text{mín}(\{90, 82\}) = 82$

Ordenamiento por Selección - Ejemplo

$\{15, 27, 48, 67, 90, 82\}$

$\text{mín}(\{67, 90, 82\}) = 67$

$\{15, 27, 48, \mathbf{67}, 90, 82\}$

$\text{mín}(\{90, 82\}) = 82$

$\{15, 27, 48, 67, \mathbf{82}, \mathbf{90}\}$

Ordenamiento por Selección - Ejemplo

$\{15, 27, 48, 67, 90, 82\}$

$\text{mín}(\{67, 90, 82\}) = 67$

$\{15, 27, 48, \mathbf{67}, 90, 82\}$

$\text{mín}(\{90, 82\}) = 82$

$\{15, 27, 48, 67, \mathbf{82}, \mathbf{90}\}$

$\text{mín}(\{90\}) = 90$

Ordenamiento por Selección - Ejemplo

$\{15, 27, 48, 67, 90, 82\}$

$\text{mín}(\{67, 90, 82\}) = 67$

$\{15, 27, 48, \mathbf{67}, 90, 82\}$

$\text{mín}(\{90, 82\}) = 82$

$\{15, 27, 48, 67, \mathbf{82}, \mathbf{90}\}$

$\text{mín}(\{90\}) = 90$

$\{15, 27, 48, 67, 82, \mathbf{90}\}$

Ordenamiento por selección - Pseudo-código

```
function ORDENAMIENTOSELECCION( $A = (a_1, \dots, a_n)$ )  
  for  $i = 1 \dots n$  do  
     $b \leftarrow \text{mín}((a_i, \dots, a_n))$   
     $A \leftarrow \text{intercambiar}(A, a_i, b)$   
  end for  
  return  $A$   
end function
```

Ordenamiento por selección (mín) - Pseudo-código

```
function mín( $A = (a_1, \dots, a_m)$ )  
   $b \leftarrow a_1$   
  for  $i = 2 \dots m$  do  
    if  $a_i < b$  then  
       $b \leftarrow a_i$   
    end if  
  end for  
  return  $b$   
end function
```

Ordenamiento por selección (mín) - Análisis

Análisis de peor caso (elemento menor en la última posición):

$$b \leftarrow a_1 \text{ (1 vez - costo } c_1)$$

Ordenamiento por selección (mín) - Análisis

Análisis de peor caso (elemento menor en la última posición):

$b \leftarrow a_1$ (1 vez - costo c_1)

$i = 1$ (1 vez - costo c_2)

Ordenamiento por selección (mín) - Análisis

Análisis de peor caso (elemento menor en la última posición):

$b \leftarrow a_1$ (1 vez - costo c_1)

$i = 1$ (1 vez - costo c_2)

comparación $a_i < b$ (m veces - costo c_3)

Ordenamiento por selección (mín) - Análisis

Análisis de peor caso (elemento menor en la última posición):

$b \leftarrow a_1$ (1 vez - costo c_1)

$i = 1$ (1 vez - costo c_2)

comparación $a_i < b$ (m veces - costo c_3)

$b \leftarrow a_i$ (m veces - costo c_4)

Ordenamiento por selección (mín) - Análisis

Análisis de peor caso (elemento menor en la última posición):

$b \leftarrow a_1$ (1 vez - costo c_1)

$i = 1$ (1 vez - costo c_2)

comparación $a_i < b$ (m veces - costo c_3)

$b \leftarrow a_i$ (m veces - costo c_4)

return (1 vez - costo c_5)

Ordenamiento por selección (mín) - Análisis

Análisis de peor caso (elemento menor en la última posición):

$b \leftarrow a_1$ (1 vez - costo c_1)

$i = 1$ (1 vez - costo c_2)

comparación $a_i < b$ (m veces - costo c_3)

$b \leftarrow a_i$ (m veces - costo c_4)

return (1 vez - costo c_5)

Costo total mín:

$$c_1 + c_2 + c_5 + m(c_3 + c_4)$$

Ordenamiento por selección (mín) - Análisis

Análisis de peor caso (elemento menor en la última posición):

$b \leftarrow a_1$ (1 vez - costo c_1)

$i = 1$ (1 vez - costo c_2)

comparación $a_i < b$ (m veces - costo c_3)

$b \leftarrow a_i$ (m veces - costo c_4)

return (1 vez - costo c_5)

Costo total mín:

$$c_1 + c_2 + c_5 + m(c_3 + c_4)$$

Simplificando:

$$d_1 + md_2$$

Ordenamiento por selección - Pseudo-código

```
function ORDENAMIENTOSELECCION( $A = (a_1, \dots, a_n)$ )  
  for  $i = 1 \dots n$  do  
     $b \leftarrow \text{mín}((a_i, \dots, a_n))$   
     $A \leftarrow \text{intercambiar}(A, a_i, b)$   
  end for  
  return  $A$   
end function
```

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

$$i = 1 \text{ (1 vez - costo } c_1)$$

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

$i = 1$ (1 vez - costo c_1)

$b \leftarrow \min((a_i, \dots, a_n))$ (n veces, la i -ésima vez cuesta

$d_1 + d_2(n - i + 1)$)

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

$i = 1$ (1 vez - costo c_1)

$b \leftarrow \min((a_i, \dots, a_n))$ (n veces, la i -ésima vez cuesta $d_1 + d_2(n - i + 1)$)

$A \leftarrow \text{intercambiar}(A, a_i, b)$ (n veces - costo c_2)

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

$i = 1$ (1 vez - costo c_1)

$b \leftarrow \min((a_i, \dots, a_n))$ (n veces, la i -ésima vez cuesta $d_1 + d_2(n - i + 1)$)

$A \leftarrow \text{intercambiar}(A, a_i, b)$ (n veces - costo c_2)

return (1 vez - costo c_3)

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

$i = 1$ (1 vez - costo c_1)

$b \leftarrow \min((a_i, \dots, a_n))$ (n veces, la i -ésima vez cuesta $d_1 + d_2(n - i + 1)$)

$A \leftarrow \text{intercambiar}(A, a_i, b)$ (n veces - costo c_2)

return (1 vez - costo c_3)

Costo total:

$$c_1 + c_3 + n(c_2 + d_1) + d_2 \sum_{i=1}^n (n - i)$$

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

Costo total:

$$c_1 + c_3 + n(c_2 + d_1) + d_2 \sum_{i=1}^n (n - i)$$

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

Costo total:

$$c_1 + c_3 + n(c_2 + d_1) + d_2 \sum_{i=1}^n (n - i)$$

Descartando constantes y re-escribiendo la sumatoria:

$$n + \sum_{i=1}^n (n - i + 1) = n + \sum_{i=1}^n i$$

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

Costo total:

$$c_1 + c_3 + n(c_2 + d_1) + d_2 \sum_{i=1}^n (n - i)$$

Descartando constantes y re-escribiendo la sumatoria:

$$\begin{aligned} n + \sum_{i=1}^n (n - i + 1) &= n + \sum_{i=1}^n i \\ &= n + \frac{n(n+1)}{2} \end{aligned}$$

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

Costo total:

$$c_1 + c_3 + n(c_2 + d_1) + d_2 \sum_{i=1}^n (n - i)$$

Descartando constantes y re-escribiendo la sumatoria:

$$\begin{aligned} n + \sum_{i=1}^n (n - i + 1) &= n + \sum_{i=1}^n i \\ &= n + \frac{n(n+1)}{2} \\ &= n + \frac{n}{2} + \frac{n^2}{2} \end{aligned}$$

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

Costo total:

$$n + \frac{n}{2} + \frac{n^2}{2}$$

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

Costo total:

$$n + \frac{n}{2} + \frac{n^2}{2}$$

Descartando constantes:

$$n + n^2$$

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

Costo total:

$$n + \frac{n}{2} + \frac{n^2}{2}$$

Descartando constantes:

$$n + n^2$$

Dejando sólo el término dominante:

$$n^2$$

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

Costo total:

$$n + \frac{n}{2} + \frac{n^2}{2}$$

Descartando constantes:

$$n + n^2$$

Dejando sólo el término dominante:

$$n^2$$

Notación O :

$$O(n^2)$$

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

$$O(n^2)$$

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

$$O(n^2)$$

Ordenar un arreglo de tamaño n con el método de ordenamiento por selección toma un tiempo $O(n^2)$

Ordenamiento por selección - Análisis

Análisis de peor caso (en cada paso elemento menor en la última posición):

$$O(n^2)$$

Ordenar un arreglo de tamaño n con el método de ordenamiento por selección toma un tiempo $O(n^2)$

Tamaño (n)	Costo (n^2)
10	100
1.000	1.000.000
100.000	10^{10}
100.000.000	10^{16}

Ordenamiento por selección - Implementación

Clase EjemploOrdenamientoSeleccion:

```
private void ordenarSeleccion(int[] arreglo) {  
    for(int izq = 0; izq < arreglo.length; izq++){  
        int der = encontrarMenor(arreglo, izq, arreglo.length);  
        intercambiarElementos(arreglo, izq, der);  
    }  
}
```

Ordenamiento por selección - Implementación

Clase EjemploOrdenamientoSeleccion:

```
private int encontrarMenor(int[] arreglo, int p1, int p2){  
    int menorIndice = p1;  
    for (int i = p1+1; i<p2; i++){  
        if(arreglo[i] < arreglo[menorIndice]) menorIndice = i;  
    }  
    return menorIndice;  
}
```

Ordenamiento por selección - Implementación

Clase EjemploOrdenamientoSeleccion:

```
private void intercambiarElementos(int[] arreglo, int p1,  
    int p2){  
    int temp = arreglo[p1];  
    arreglo[p1] = arreglo[p2];  
    arreglo[p2] = temp;  
}
```

Ordenamiento por Inserción

Dado un arreglo $A = (a_1, \dots, a_n)$

Ordenamiento por Inserción

Dado un arreglo $A = (a_1, \dots, a_n)$

Primera posición aislada siempre está ordenada $((a_1))$

Ordenamiento por Inserción

Dado un arreglo $A = (a_1, \dots, a_n)$

Primera posición aislada siempre está ordenada $((a_1))$

Tomar el *segundo* elemento y moverlo a la izquierda hasta que las *dos primeras* posiciones estén ordenadas $((a_1, a_2))$

Ordenamiento por Inserción

Dado un arreglo $A = (a_1, \dots, a_n)$

Primera posición aislada siempre está ordenada $((a_1))$

Tomar el *segundo* elemento y moverlo a la izquierda hasta que las *dos primeras* posiciones estén ordenadas $((a_1, a_2))$

Tomar el *tercer* elemento y moverlo a la izquierda hasta que las *tres primeras* posiciones estén ordenadas $((a_1, a_2, a_3))$

Ordenamiento por Inserción

Dado un arreglo $A = (a_1, \dots, a_n)$

Primera posición aislada siempre está ordenada $((a_1))$

Tomar el *segundo* elemento y moverlo a la izquierda hasta que las *dos primeras* posiciones estén ordenadas $((a_1, a_2))$

Tomar el *tercer* elemento y moverlo a la izquierda hasta que las *tres primeras* posiciones estén ordenadas $((a_1, a_2, a_3))$

Repetir con todos los elementos hasta el n -ésimo $((a_1, a_2, \dots, a_n))$

Ordenamiento por Inserción - Pseudo-código

```
function ORDENAMIENTOINSERCIÓN( $A = (a_1, \dots, a_n)$ )  
  for  $i = 2 \dots, n$  do  
     $b \leftarrow a_i$   
     $j \leftarrow i - 1$   
    while  $j > 0$  and  $a_j > b$  do  
       $a_{j+1} \leftarrow a_j$   
       $j \leftarrow j - 1$   
    end while  
     $a_{j+1} \leftarrow b$   
  end for  
  return  $A$   
end function
```

Ordenamiento por Inserción - Ejemplo

{82, 15, 67, 27, 90, 48}

Ordenamiento por Inserción - Ejemplo

$\{82, 15, 67, 27, 90, 48\}$

$\{82\}$ ya ordenada

Ordenamiento por Inserción - Ejemplo

$\{82, 15, 67, 27, 90, 48\}$

$\{82\}$ ya ordenada

Seleccionamos $a_2 = 15$ y lo movemos a la izquierda hasta que las dos primeras posiciones estén ordenadas

$\{15, 82, 67, 27, 90, 48\}$

Ordenamiento por Inserción - Ejemplo

$\{82, 15, 67, 27, 90, 48\}$

$\{82\}$ ya ordenada

Seleccionamos $a_2 = 15$ y lo movemos a la izquierda hasta que las dos primeras posiciones estén ordenadas

$\{15, 82, 67, 27, 90, 48\}$

$\{15, 82\}$ ya ordenada

Ordenamiento por Inserción - Ejemplo

$\{82, 15, 67, 27, 90, 48\}$

$\{82\}$ ya ordenada

Seleccionamos $a_2 = 15$ y lo movemos a la izquierda hasta que las dos primeras posiciones estén ordenadas

$\{15, 82, 67, 27, 90, 48\}$

$\{15, 82\}$ ya ordenada

Seleccionamos $a_3 = 67$ y lo movemos a la izquierda hasta que las 3 primeras posiciones estén ordenadas

$\{15, 67, 82, 27, 90, 48\}$

Ordenamiento por Inserción - Ejemplo

$\{82, 15, 67, 27, 90, 48\}$

$\{82\}$ ya ordenada

Seleccionamos $a_2 = 15$ y lo movemos a la izquierda hasta que las dos primeras posiciones estén ordenadas

$\{15, 82, 67, 27, 90, 48\}$

$\{15, 82\}$ ya ordenada

Seleccionamos $a_3 = 67$ y lo movemos a la izquierda hasta que las 3 primeras posiciones estén ordenadas

$\{15, 67, 82, 27, 90, 48\}$

$\{15, 67, 82\}$ ya ordenada

Ordenamiento por Inserción - Ejemplo

{15, **67**, 82, 27, 90, 48}

Ordenamiento por Inserción - Ejemplo

$\{15, \mathbf{67}, 82, 27, 90, 48\}$

Seleccionamos $a_4 = 27$ y lo movemos a la izquierda hasta que las 4 primeras posiciones estén ordenadas

$\{15, \mathbf{27}, 67, 82, 90, 48\}$

Ordenamiento por Inserción - Ejemplo

$\{15, \mathbf{67}, 82, 27, 90, 48\}$

Seleccionamos $a_4 = 27$ y lo movemos a la izquierda hasta que las 4 primeras posiciones estén ordenadas

$\{15, \mathbf{27}, 67, 82, 90, 48\}$

$\{15, 27, 67, 82\}$ ya ordenada

Ordenamiento por Inserción - Ejemplo

$\{15, \mathbf{67}, 82, 27, 90, 48\}$

Seleccionamos $a_4 = 27$ y lo movemos a la izquierda hasta que las 4 primeras posiciones estén ordenadas

$\{15, \mathbf{27}, 67, 82, 90, 48\}$

$\{15, 27, 67, 82\}$ ya ordenada

Seleccionamos $a_5 = 90$ y lo movemos a la izquierda hasta que las 5 primeras posiciones estén ordenadas

$\{15, 27, 67, 82, \mathbf{90}, 48\}$

Ordenamiento por Inserción - Ejemplo

$\{15, \mathbf{67}, 82, 27, 90, 48\}$

Seleccionamos $a_4 = 27$ y lo movemos a la izquierda hasta que las 4 primeras posiciones estén ordenadas

$\{15, \mathbf{27}, 67, 82, 90, 48\}$

$\{15, 27, 67, 82\}$ ya ordenada

Seleccionamos $a_5 = 90$ y lo movemos a la izquierda hasta que las 5 primeras posiciones estén ordenadas

$\{15, 27, 67, 82, \mathbf{90}, 48\}$

$\{15, 27, 67, 82, 90\}$ ya ordenada

Ordenamiento por Inserción - Ejemplo

$\{15, \mathbf{67}, 82, 27, 90, 48\}$

Seleccionamos $a_4 = 27$ y lo movemos a la izquierda hasta que las 4 primeras posiciones estén ordenadas

$\{15, \mathbf{27}, 67, 82, 90, 48\}$

$\{15, 27, 67, 82\}$ ya ordenada

Seleccionamos $a_5 = 90$ y lo movemos a la izquierda hasta que las 5 primeras posiciones estén ordenadas

$\{15, 27, 67, 82, \mathbf{90}, 48\}$

$\{15, 27, 67, 82, 90\}$ ya ordenada

Seleccionamos $a_6 = 48$ y lo movemos a la izquierda hasta que las 6 primeras posiciones estén ordenadas

$\{15, 27, \mathbf{48}, 67, 82, 90\}$

Ordenamiento por Inserción - Pseudo-código

```
function ORDENAMIENTOINSERCIÓN( $A = (a_1, \dots, a_n)$ )  
  for  $i = 2 \dots, n$  do  
     $b \leftarrow a_i$   
     $j \leftarrow i - 1$   
    while  $j > 0$  and  $a_j > b$  do  
       $a_{j+1} \leftarrow a_j$   
       $j \leftarrow j - 1$   
    end while  
     $a_{j+1} \leftarrow b$   
  end for  
  return  $A$   
end function
```


Ordenamiento por Inserción - Análisis

Análisis de peor caso (en cada paso elemento seleccionado debe moverse hasta el inicio - arreglo ordenado al revés):

$$i = 2 \text{ (1 vez - costo } c_1)$$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (en cada paso elemento seleccionado debe moverse hasta el inicio - arreglo ordenado al revés):

$i = 2$ (1 vez - costo c_1)

$b \leftarrow a_i$ ($n - 1$ veces - costo c_2)

Ordenamiento por Inserción - Análisis

Análisis de peor caso (en cada paso elemento seleccionado debe moverse hasta el inicio - arreglo ordenado al revés):

$$i = 2 \text{ (1 vez - costo } c_1)$$

$$b \leftarrow a_i \text{ (} n - 1 \text{ veces - costo } c_2)$$

$$j \leftarrow i - 1 \text{ (} n - 1 \text{ veces - costo } c_3)$$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (en cada paso elemento seleccionado debe moverse hasta el inicio - arreglo ordenado al revés):

$i = 2$ (1 vez - costo c_1)

$b \leftarrow a_i$ ($n - 1$ veces - costo c_2)

$j \leftarrow i - 1$ ($n - 1$ veces - costo c_3)

Evaluar $j > 0$ y $a_j > b$ ($n - 1$ veces, la i -ésima vez i veces - costo c_4)

Ordenamiento por Inserción - Análisis

Análisis de peor caso (en cada paso elemento seleccionado debe moverse hasta el inicio - arreglo ordenado al revés):

$i = 2$ (1 vez - costo c_1)

$b \leftarrow a_i$ ($n - 1$ veces - costo c_2)

$j \leftarrow i - 1$ ($n - 1$ veces - costo c_3)

Evaluar $j > 0$ y $a_j > b$ ($n - 1$ veces, la i -ésima vez i veces - costo c_4)

$a_{j+1} \leftarrow a_j$ y $j \leftarrow j + 1$ ($n - 1$ veces, la i -ésima vez i veces - costo c_5)

Ordenamiento por Inserción - Análisis

Análisis de peor caso (en cada paso elemento seleccionado debe moverse hasta el inicio - arreglo ordenado al revés):

$i = 2$ (1 vez - costo c_1)

$b \leftarrow a_i$ ($n - 1$ veces - costo c_2)

$j \leftarrow i - 1$ ($n - 1$ veces - costo c_3)

Evaluar $j > 0$ y $a_j > b$ ($n - 1$ veces, la i -ésima vez i veces - costo c_4)

$a_{j+1} \leftarrow a_j$ y $j \leftarrow j + 1$ ($n - 1$ veces, la i -ésima vez i veces - costo c_5)

$a_{j+1} \leftarrow b$ ($n - 1$ veces - costo c_6)

Costo total:

$$c_1 + (c_2 + c_3 + c_6)n + (c_4 + c_5) \sum_{i=2}^n i$$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (en cada paso elemento seleccionado debe moverse hasta el inicio - arreglo ordenado al revés):

Costo total:

$$c_1 + (c_2 + c_3 + c_6)n + (c_4 + c_5) \sum_{i=2}^n i$$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (en cada paso elemento seleccionado debe moverse hasta el inicio - arreglo ordenado al revés):

Costo total:

$$c_1 + (c_2 + c_3 + c_6)n + (c_4 + c_5) \sum_{i=2}^n i$$

Descartando constantes y re-escribiendo la sumatoria:

$$n + \sum_{i=2}^n i = n + \sum_{i=1}^{n-1} (i + 1)$$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (en cada paso elemento seleccionado debe moverse hasta el inicio - arreglo ordenado al revés):

Costo total:

$$c_1 + (c_2 + c_3 + c_6)n + (c_4 + c_5) \sum_{i=2}^n i$$

Descartando constantes y re-escribiendo la sumatoria:

$$\begin{aligned} n + \sum_{i=2}^n i &= n + \sum_{i=1}^{n-1} (i + 1) \\ &= n + \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 \end{aligned}$$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (en cada paso elemento seleccionado debe moverse hasta el inicio - arreglo ordenado al revés):

Costo total:

$$c_1 + (c_2 + c_3 + c_6)n + (c_4 + c_5) \sum_{i=2}^n i$$

Descartando constantes y re-escribiendo la sumatoria:

$$\begin{aligned} n + \sum_{i=2}^n i &= n + \sum_{i=1}^{n-1} (i + 1) \\ &= n + \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 \\ &= n + \frac{(n-1)n}{2} + n - 1 \end{aligned}$$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (arreglo ordenado al revés):

Costo total:

$$2n - 1 + \frac{(n - 1)n}{2}$$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (arreglo ordenado al revés):

Costo total:

$$2n - 1 + \frac{(n-1)n}{2}$$

Descartando constantes:

$$n + n^2$$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (arreglo ordenado al revés):

Costo total:

$$2n - 1 + \frac{(n-1)n}{2}$$

Descartando constantes:

$$n + n^2$$

Dejando sólo el término dominante:

$$n^2$$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (arreglo ordenado al revés):

Costo total:

$$2n - 1 + \frac{(n - 1)n}{2}$$

Descartando constantes:

$$n + n^2$$

Dejando sólo el término dominante:

$$n^2$$

Notación O :

$$O(n^2)$$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (arreglo ordenado al revés):

$$O(n^2)$$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (arreglo ordenado al revés):

$$O(n^2)$$

Ordenar un arreglo de tamaño n con el método de ordenamiento por inserción toma un tiempo $O(n^2)$

Ordenamiento por Inserción - Análisis

Análisis de peor caso (arreglo ordenado al revés):

$$O(n^2)$$

Ordenar un arreglo de tamaño n con el método de ordenamiento por inserción toma un tiempo $O(n^2)$

Tamaño (n)	Costo (n^2)
10	100
1.000	1.000.000
100.000	10^{10}
100.000.000	10^{16}

Resumen

Algoritmos de búsqueda:

Resumen

Algoritmos de búsqueda: Búsqueda Lineal

Resumen

Algoritmos de búsqueda:

Búsqueda Lineal

Búsqueda Binaria

Resumen

Algoritmos de búsqueda:

Búsqueda Lineal

Búsqueda Binaria

Algoritmos de ordenamiento:

Resumen

Algoritmos de búsqueda:

- Búsqueda Lineal

- Búsqueda Binaria

Algoritmos de ordenamiento:

- Ordenamiento por selección

Resumen

Algoritmos de búsqueda:

- Búsqueda Lineal

- Búsqueda Binaria

Algoritmos de ordenamiento:

- Ordenamiento por selección

- Ordenamiento por inserción

Resumen

Algoritmos de búsqueda:

- Búsqueda Lineal

- Búsqueda Binaria

Algoritmos de ordenamiento:

- Ordenamiento por selección

- Ordenamiento por inserción

Análisis de costo computacional