



Diseño de Algoritmos

Sesión 06

Profesores:

Tomás Lara Valdovinos – sir.thomas.lara@gmail.com

Jessica Meza-Jaque – jessicamezajaque.uchile@gmail.com

OBJETIVOS DE LA SESIÓN

- Conocer algoritmos iterativos y recursivos
- Identificar diferencias entre ellos
- Reflexionar sobre la eficiencia de los algoritmos iterativos respecto de los recursivos



CONTENIDOS DE LA SESIÓN



- Algoritmos iterativos de Ordenamiento por burbuja y búsqueda binaria.
- Algoritmos recursivos de búsqueda binaria y cálculo de factorial.
- DESAFÍO 03

Algoritmos Iterativos

- Resuelven problemas
- Ejecutan una actividad de manera continuada
- Hasta transformar el entorno en la solución

A cartoon illustration of a man with blue hair, wearing a black suit, white shirt, and grey tie. He is standing in a kitchen, looking angry or stressed, with his hands on his hips. He is surrounded by a messy kitchen scene. In the foreground, there is a sink with a faucet, a glass of water, and a small container. To the left of the sink, there is a large glass pitcher filled with brown liquid, a bowl of green grapes, and a stack of plates. To the right of the sink, there is a stack of plates, a bowl of pink liquid, and a small container. In the background, there is a stove with a pot on it, a stack of plates, and a small container. The kitchen has white tiled walls and a window with a blue frame.

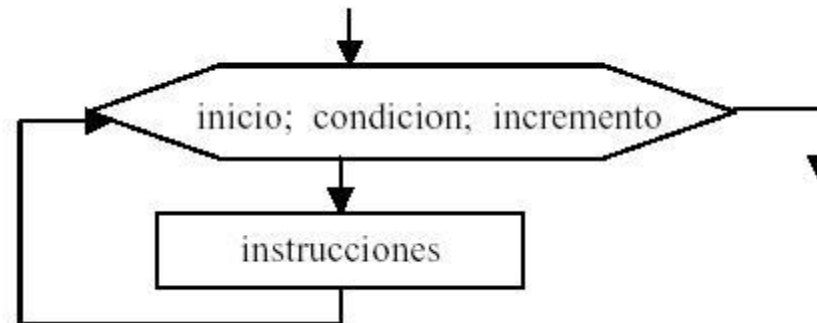
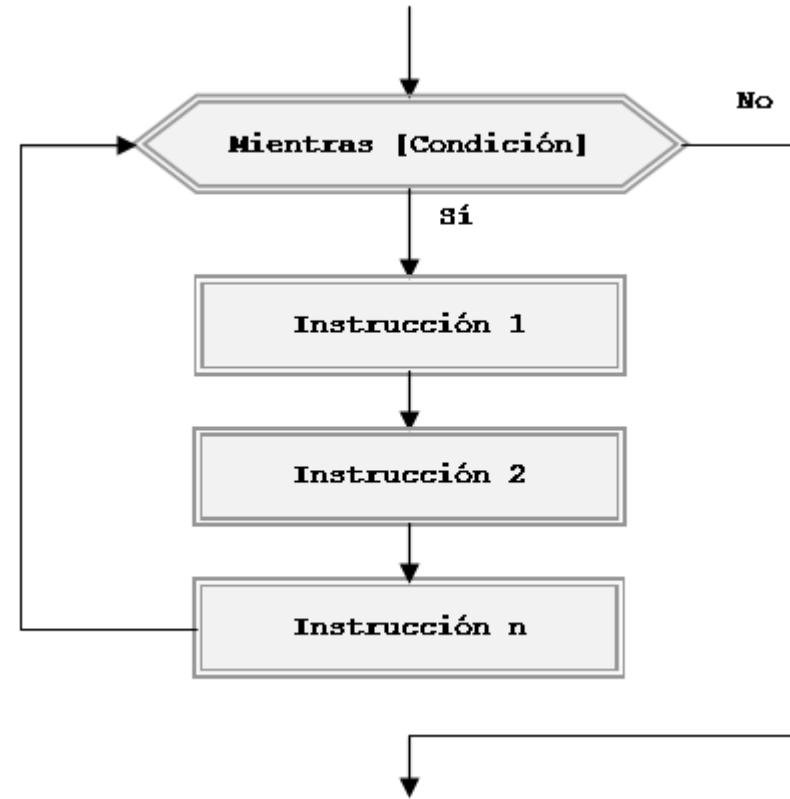
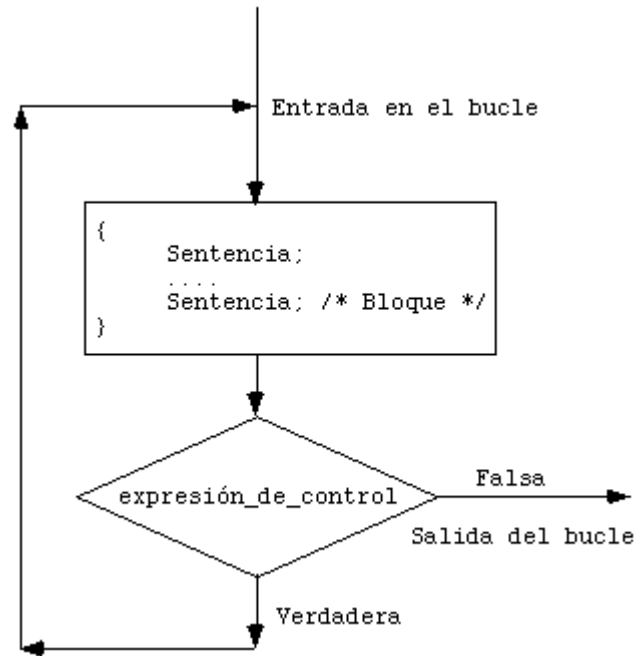
- Situación: Lavar todas las piezas de loza
- Posible solución: Utilizar un procedimiento (secuencia de instrucciones) que itere sobre cada pieza y permita limpiarlas.

Ciclos

- Para definir algoritmos iterativos utilizamos los ciclos.
- En un lenguaje de programación utilizamos por ejemplo:
 - While
 - For
 - Do While



Ciclos



Ejemplo – Lavar los platos

```
lavarLoza(Loza)
    while queda_sucia(Loza) do
        lavar(Loza.next())
    End
```

```
lavarLozaFor(Loza)
    for i = 0 to Loza.length do
        lavar(Loza[i])
    End
```


Complejidad de algoritmos iterativos

- Definimos como **complejidad de un algoritmo iterativo** como la cantidad de iteraciones que deben ejecutarse las actividades en éste.



Ordenamiento de burbuja

Mejor	Promedio	Peor
$O(n^2)$	$O(n^2)$	$O(n^2)$

Ordenar(A)

 For i = 1 to n do

 For j=0 to n-i do

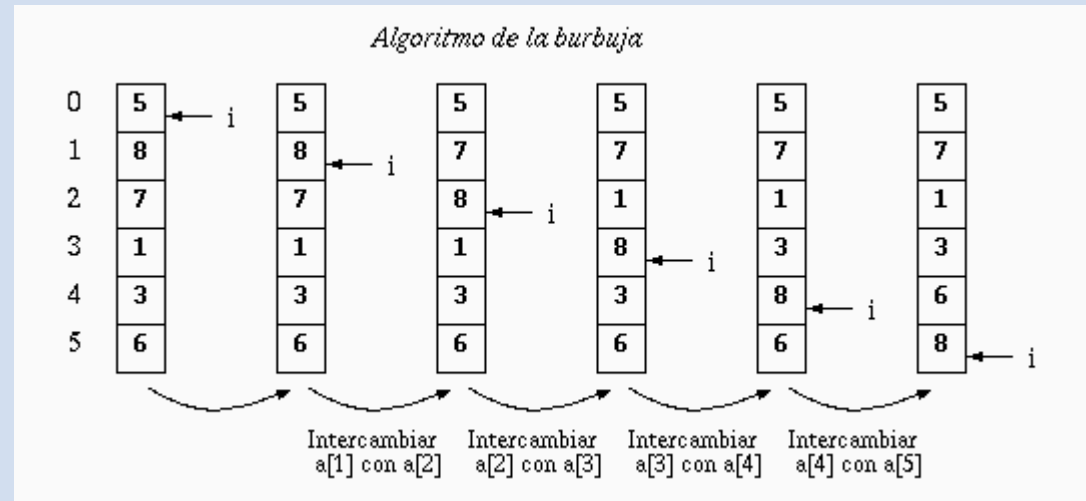
 If $A[j] > A[j+1]$ do

 Aux = A[j]

 A[j] = A[j+1]

 A[j+1] = Aux

end



Búsqueda Binaria

Mejor	Promedio	Peor
O(1)	O(log n)	O(log n)

```
Buscar(A, t)
  low = 0
  high = n-1
  while low <= high do
    ix = (low + high)/2
    if t = A[ix] then
      return true
    else if t < A[ix] then
      high = ix - 1
    else low = ix + 1
  return false
End
```

Buscar (A, 11)

	low		ix		high		
Primera Iteración	1	4	8	9	11	15	17

			low		ix	high	
Segunda Iteración	1	4	8	9	11	15	17

			low		ix	high	
Tercera Iteración	1	4	8	9	11	15	17

Elementos Explorados ($\sim \log_2 n$)

Recursividad

- Resolver un problema mediante **recursión** significa que la solución depende de las soluciones de pequeñas instancias del mismo problema.
- Decimos que un problema puede ser resuelto mediante la **recursividad** si podemos encontrar las instancias del problema que pueden ser solucionadas directamente, a esto lo llamamos **Caso base**.



Ejemplo - Matrioska

- ¿Alguna vez has visto un juego de muñecas rusas? Al principio, solo ves una figurilla, generalmente de madera pintada, que se ve más o menos así:



Ejemplo - Matrioska

- Puedes quitar la mitad de arriba de la primera muñeca, ¿y qué ves adentro? ¡Otra muñeca rusa, un poco más pequeña!



Ejemplo - Matrioska

- Y puedes continuar. Eventualmente encontrarás a la muñeca rusa más pequeña. Es solo una pieza, así que no se abre:



Empezamos con un muñeca rusa grande, y vimos muñecas rusas más y más pequeñas, hasta que vimos una que era tan pequeña que no podría contener a otra.

Llamadas recursivas a funciones

- Podemos implementar la **recursividad utilizando una llamada de un método o función hacia sí misma.**

```
function recursividad(subproblema){  
    if (es_caso_base(subproblema)){  
        return solución_base(subproblema);  
    }  
    recursividad(descomponer(subproblema));  
}
```



Ejemplo - Matrioska

```
function abrir_matrioska(A, muneca){  
    A.add(muneca);  
    if(!muneca.sePuedeAbrir()){  
        return;  
    }  
    abrir_matrioska(muneca.hija);  
}
```

Complejidad de algoritmos recursivos

- Se mide como la cantidad de llamadas recursivas necesarias antes de llegar al caso base.
- La complejidad espacial es importante cuando usamos algoritmos recursivos.



Búsqueda Binaria			
Mejor	Promedio	Peor	
O(1)	O(log n)	O(log n)	

```
Buscar(low, high, A, t)
    ix = (low + high)/2
    if t = A[ix] then
        return ix
    else if t < A[ix] then
        return Buscar(low, ix - 1, A, t)
    else
        return Buscar(ix + 1, high, A, t)
End
```

Cálculo del factorial forma recursiva

$$\text{fact}(n) = \begin{cases} \text{si } n = 0 & \longrightarrow 1 \\ \text{si } n > 0 & \longrightarrow n \cdot \text{fact}(n - 1) \end{cases}$$

función factorial:

input: entero n de forma que $n \geq 0$

output: $[n \times (n-1) \times (n-2) \times \dots \times 1]$

1. **if** n es 0, **return** 1
2. **else**, **return** $[n \times \text{factorial}(n-1)]$

end factorial

Desafío 03

- ***Busque o implemente*** el algoritmo de ordenamiento **QuickSort** en sus formas iterativas y recursivas en un lenguaje de programación a elección.
- Explique las **diferencias notables** entre las 2 implementaciones.
- Realice pruebas de funcionamiento y **compare** quicksort (Iterativo y Recursivo) con ordenamiento por inserción para **distintos valores de n**.



Desafío 03

Se espera que usted entregue:

- Una página, **tipo Poster**, con la siguiente estructura:
 - Un título creativo, que llame la atención del lector, y que esté referido al resultado y/o sus hallazgos
 - Un breve desarrollo de dichos hallazgos
 - Su propio aporte de valor (considera usted que sus hallazgos tienen alguna implicancia en su vida como estudiante de ICINF y/o como futuro profesional? Por qué?
 - Referencias bibliográficas

Nota: Intente construir este poster de una forma lo más gráfica posible (más imágenes, menos texto)
- Dos páginas de anexos con los programas que usted construyó y que avalan sus resultados.
- Procedimiento que utilizó para llevar a cabo las pruebas, p.e. usando gráficas, comparaciones matemáticas, etc. Además explique los resultados de las pruebas de comparación.

Desafío 03

Plazo de entrega



Domingo 10 de Abril de 2016
hasta las 23:55hrs en aula virtual
de nuestro curso.

CHECK - OBJETIVOS DE LA SESIÓN

- Conocer algoritmos iterativos y recursivos
- Identificar diferencias entre ellos
- Reflexionar sobre la eficiencia de los algoritmos iterativos respecto de los recursivos

CHECK





Diseño de Algoritmos

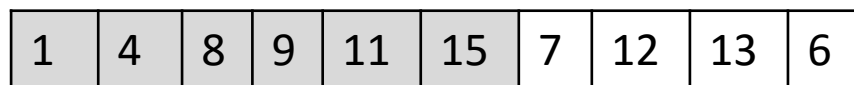
Sesión 06

Profesores:

Tomás Lara Valdovinos – sir.thomas.lara@gmail.com

Jessica Meza-Jaque – jessicamezajaque.uchile@gmail.com

Inserción (A, 6, 7)



Ya ordenado

Insertamos en el punto
que corresponde

7 Valor

Se mueven los elementos que estaban
ordenados hacia adelante




Región ordenada se extiende por uno

Buscar (A, 11)

	low		ix		high		
Primera Iteración	1	4	8	9	11	15	17

	low		ix		high		
Segunda Iteración	1	4	8	9	11	15	17

	low		ix		high		
Tercera Iteración	1	4	8	9	11	15	17



Elementos Explorados ($\sim \log_2 n$)