Diseño de Algoritmos

Sesión 07

Profesores:

Tomás Lara Valdovinos – t.lara@uandresbello.edu

Jessica Meza-Jaque – je.meza@uandresbello.edu

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.

Algoritmos Iterativos

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

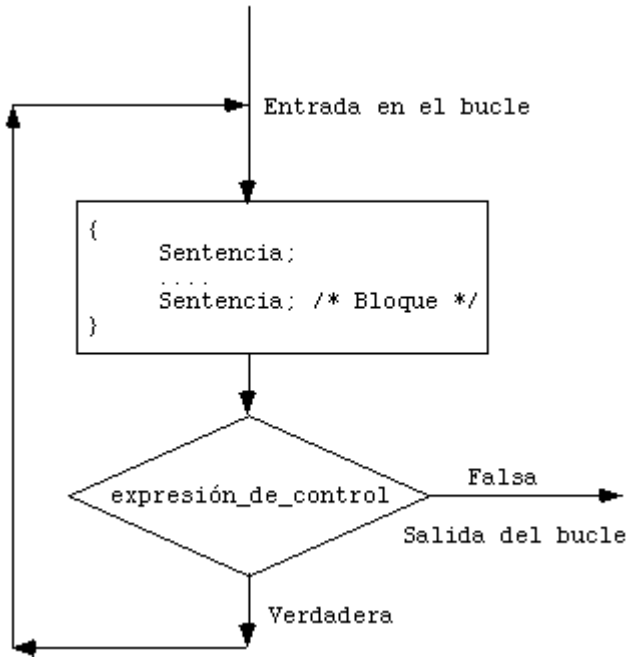
Ejemplo – Lavar la loza

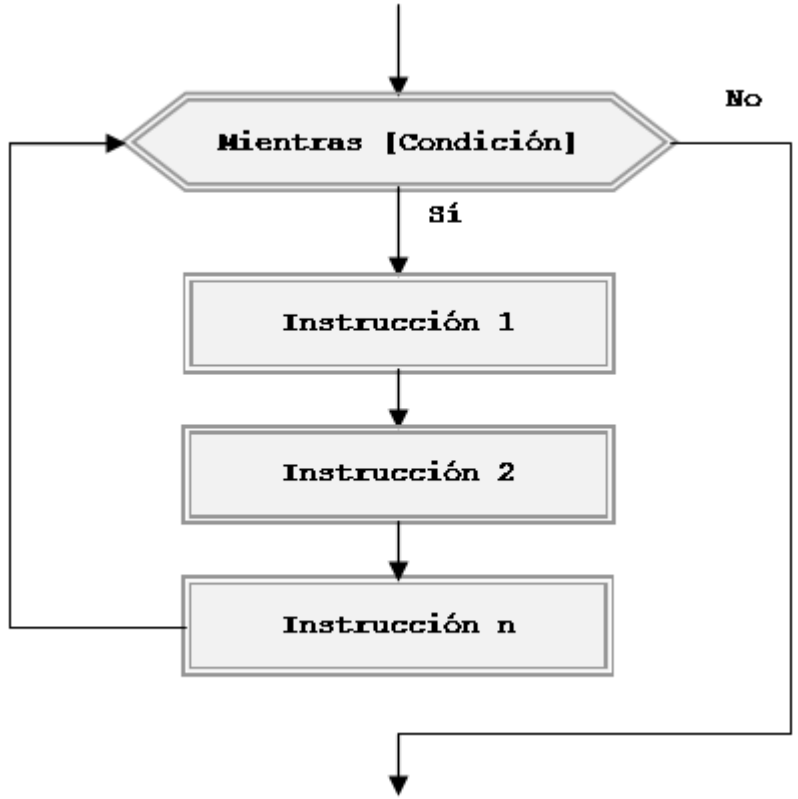
- Situación: Lavar todas las piezas de loza
- Posible solución: Utilizar un procedimiento (secuencia de instrucciones) que itere sobre cada pieza y permita limpiarlas.

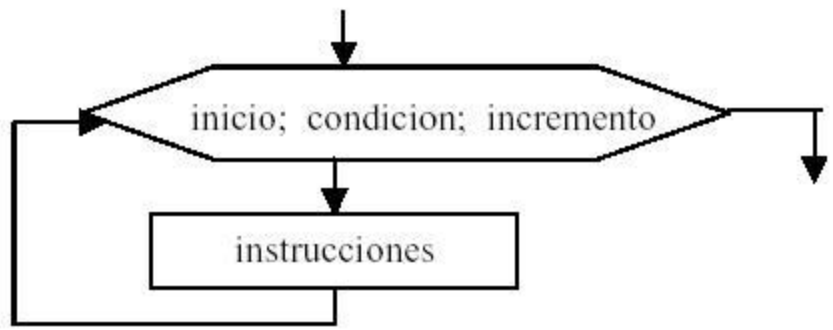
Ciclos

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

Ciclos


Entrada en el bucle
{
Sentencia;
....
Sentencia; /* Bloque */
}
expresión_de_control
Falsa
Salida del bucle
Verdadera


Mientras [Condición]
No
Sí
Instrucción 1
Instrucción 2
Instrucción n


inicio; condicion; incremento
instrucciones

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
```

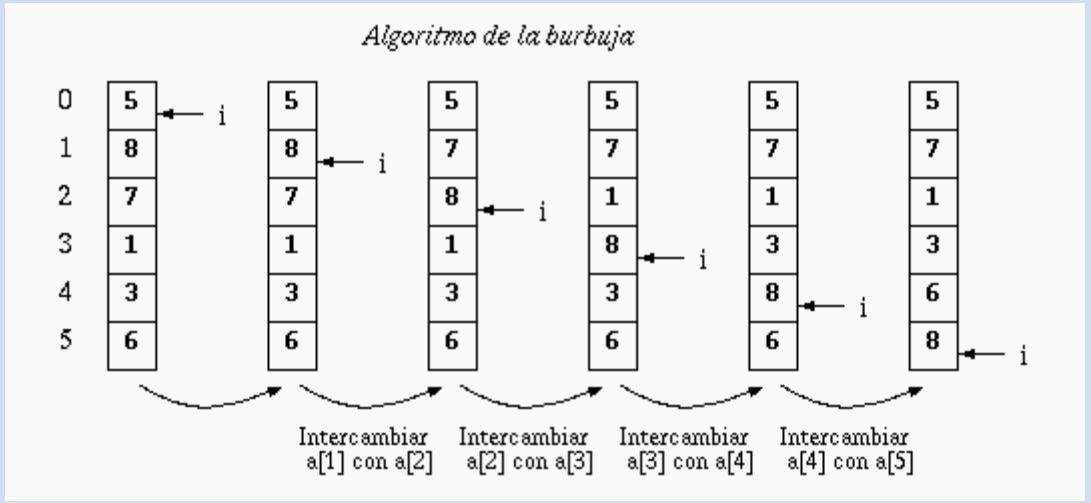
Complejidad de algoritmos iterativos

- 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
$O(n^2)$	$O(n^2)$	$O(n^2)$

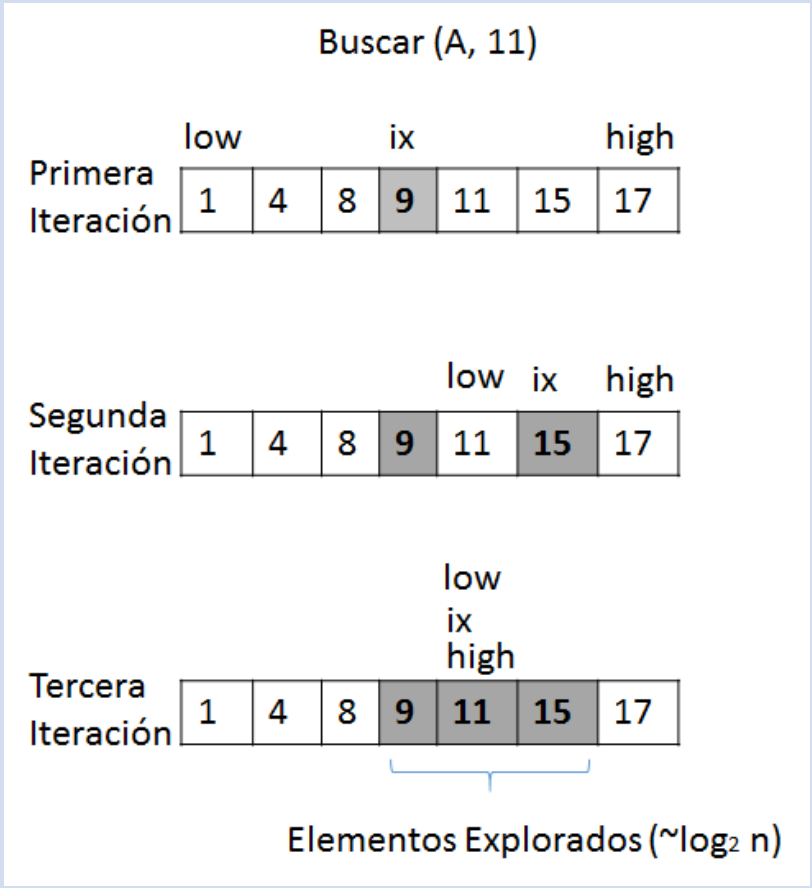
```
Ordenar(A)
    For i = 1 to n do
        For j=0 to n-i do
            If A[j] > A[j+1] do
                Aux = A[j]
                A[j] = A[j+1]
                A[j+1] = Aux
end
```



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
```



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(!muñeca.sePuedeAbrir()){
		return;
	}
	abrir_matrioska(muñeca.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

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

```
función factorial:
input: entero n de forma que n >= 0
output: [n × (n-1) × (n-2) × … × 1]

    1. if n es 0, return 1
    2. else, return [ n × factorial(n-1) ]

end factorial
```

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 07

Profesores:

Tomás Lara Valdovinos – t.lara@uandresbello.edu

Jessica Meza-Jaque – je.meza@uandresbello.edu