



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**

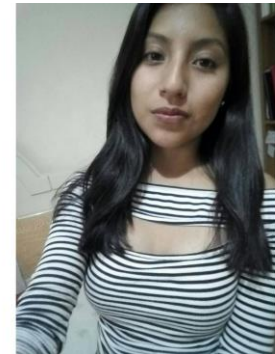


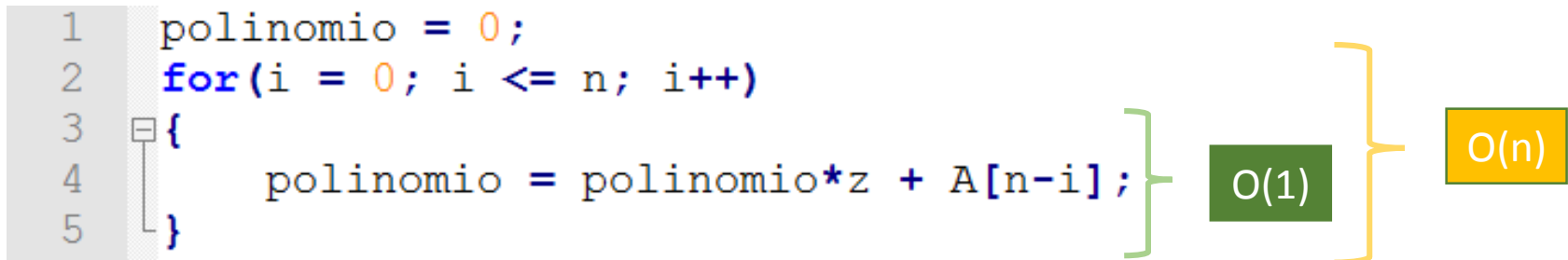
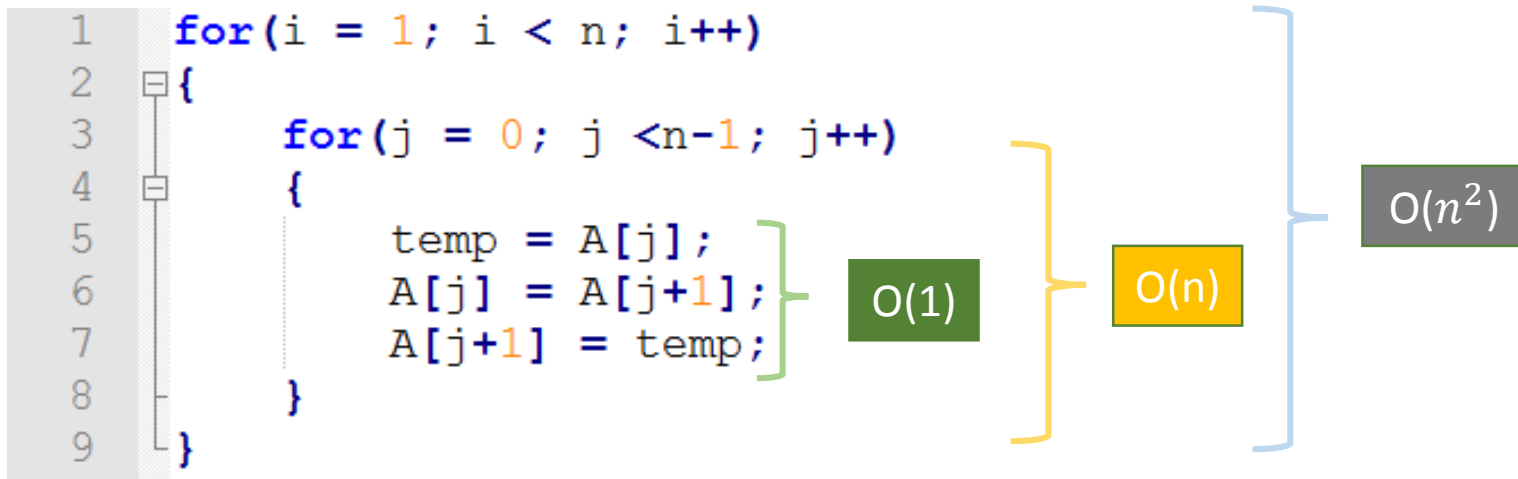
Análisis de algoritmos

**Ejercicio 04 :
Análisis de algoritmos no recursivos**

**Alumno(a):
Abigail Nicolás Sayago**

Grupo: 3CM3





```
1  for(i = 1; i <= n; i++)
2      for(j = 1; j <= n; j++)
3          C[i,j] = 0;
4          for(k = 1; k <= n; k++)
5              C[i,j] = C[i,j] + A[i,k]*B[k,j];
6
```

$O(n)$

$O(n^2)$

$O(n^3)$

```
1  anterior = 1;
2  actual = 1;
3  while(n > 2)
4  {
5      aux = anterior + actual;
6      anterior = actual;
7      actual = aux;
8      n = n-1;
9  }
```

$O(1)$

$O(n^2)$

```
1  for(i = n - 1; j = 0; i >= 0; i--, j++)  
2      s2[j] = s[i];  
3  
4  for(i = 0; i < n; i++)  
5      s[i] = s2[i];  
6
```

$O(1)$ $O(n)$

$O(1)$ $O(n)$

```
1 func Producto2Mayores(A, n)
2     if (A[1] > A[2])
3         mayor1 = A[1];
4         mayor2 = A[2];
5     else
6         mayor1 = A[2];
7         mayor2 = A[1];
8
9     i = 3;
10
11     while(i <= n)
12         if(A[i] > mayor1)
13             mayor2 = mayor1;
14             mayor1 = A[i];
15         else if (A[i] > mayor2)
16             mayor2 = A[i];
17
18         i = i + 1;
19     return mayor1 * mayor2;
```

Complexity analysis annotations:

- Lines 2-4: $O(1)$
- Lines 5-7: $O(1)$
- Lines 12-14: $O(1)$
- Lines 15-17: $O(1)$
- Lines 12-17 (entire while loop body): $O(n)$

```

1 func OrdenamientoIntercambio(a, n)
2   for(i = 0; i < n-1; i++)
3     for(int j = i + 1; j < n; j++)
4       if(a[j] < a[i])
5         { temp = a[i];
6           a[i] = a[j];
7           a[j] = temp;
8         }
9   fin

```

$O(1)$

$O(n)$

$O(n^2)$

```

1 func MaximoComunDivisor(m, n)
2   {
3     a = max(n, m);
4     b = min (n, m);
5     residuo = 1;
6     while(residuo > 0)
7     {
8       residuo = a mod b;
9       a = b;
10      b = residuo;
11    }
12    MaximoComunDivisor = a;
13    return MaximoComunDivisor;
14  }

```

$O(1)$

$O(\log n)$

```

1  Procedimiento BurbujaOptimizada(A, n)
2      cambios = "No";
3      i = 0;
4      while((i < n-1) && (cambios != "No"))
5      {
6          cambios = "No";
7          for(j=0; j<=(n-2)-i; j++)
8          {
9              if (A[j] > A[j+1])
10             {
11                 aux = A[j];
12                 A[j] = A[j+1];
13                 A[j+1] = aux;
14                 cambios = "Si";
15             }
16         }
17         i++;
18     }
19  Fin Procedimiento

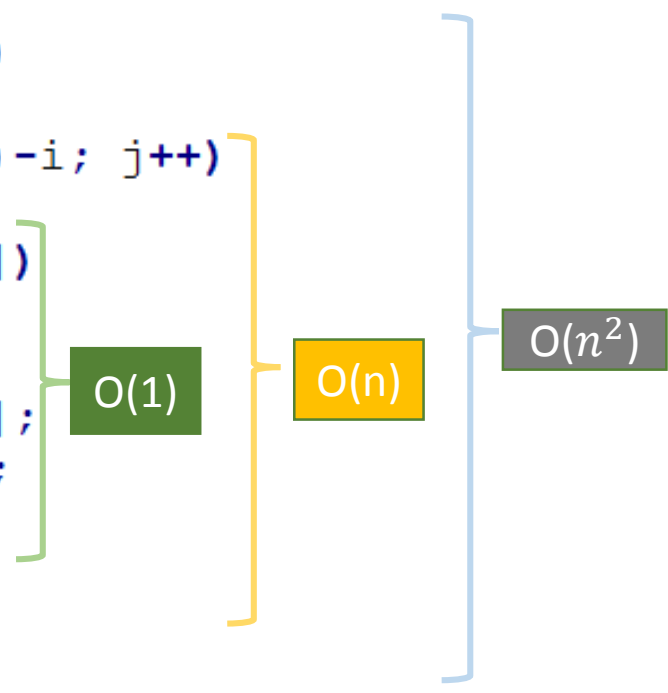
```

$O(1)$

$O(n)$

$O(n^2)$

```
1  Procedimiento BurbujaSimple(A, n)
2  {
3      for(i = 0; i <= n-2; i++)
4      {
5          for(j = 0; j <= (n-2)-i; j++)
6          {
7              if (A[j] > A[j+1])
8              {
9                  aux = A[j];
10                 A[j] = A[j+1];
11                 A[j+1] = aux;
12             }
13         }
14     }
15 }
16 Fin Procedimiento
```



The diagram illustrates the time complexity of the provided code. It uses nested brackets to group operations and their respective complexities:

- A green bracket groups the innermost loop body (lines 7-12), which is labeled with a green box containing $O(1)$.
- A yellow bracket groups the entire inner loop (lines 5-13), which is labeled with a yellow box containing $O(n)$.
- A blue bracket groups the entire function body (lines 3-15), which is labeled with a grey box containing $O(n^2)$.

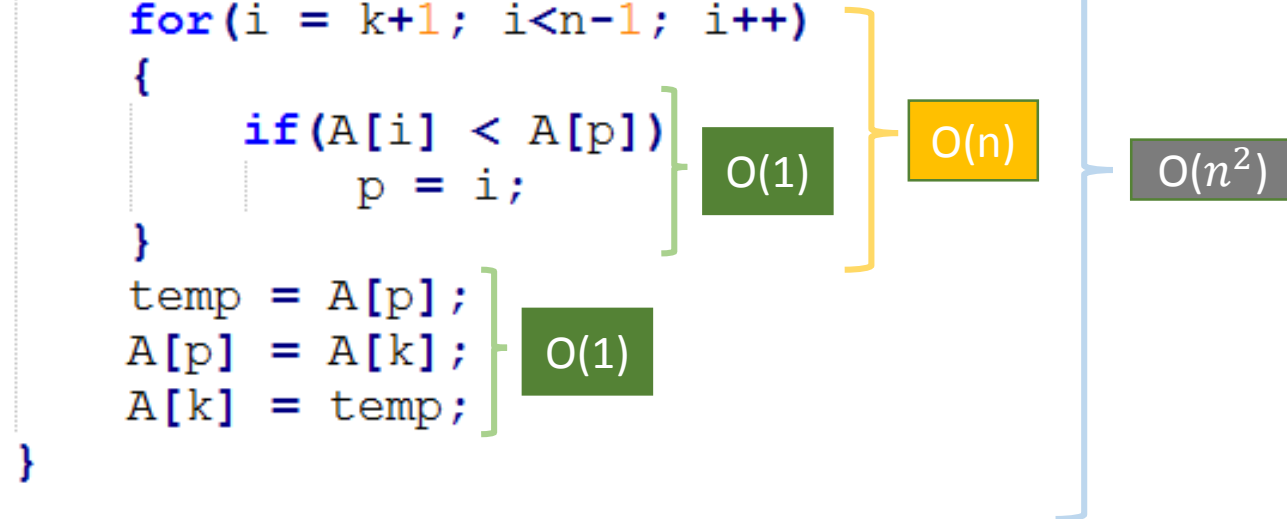

```

1  Procedimiento Ordena(a, b, c)
2  {
3      if(a > b) } O(1)
4          if(a > c)
5              if(b > c)
6                  salida (a, b, c);
7              else
8                  salida (a, c, b);
9          else
10             salida (c, a, b);
11     else
12         if(b > c) } O(1)
13         if(a > c) } O(1)
14             salida(b, a, c);
15         else
16             salida(b, c, a); } O(1)
17     else
18         salida(c, b, a);
19 }

```

El peor caso es cuando se deben ejecutar 3 operaciones constantes y 3 comparaciones.

```
1  Procedimiento Seleccion(A, n)
2  {
3      for(k=0; k<n-2; k++)
4      {
5          p=k;
6          for(i = k+1; i<n-1; i++)
7          {
8              if(A[i] < A[p])
9                  p = i;
10         }
11         temp = A[p];
12         A[p] = A[k];
13         A[k] = temp;
14     }
15 }
16 Fin Procedimiento
```



The diagram illustrates the time complexity of the Selection Sort algorithm. It uses nested curly braces to group code blocks and assign them complexity values in colored boxes:

- A green brace groups the innermost `if` statement (lines 8-9), labeled with a green box containing $O(1)$.
- An orange brace groups the inner `for` loop (lines 6-9), labeled with an orange box containing $O(n)$.
- A blue brace groups the entire function body (lines 3-14), labeled with a grey box containing $O(n^2)$.