# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO


## FACULTAD DE ESTUDIOS SUPERIORES ARAGON

# TAREA

**P R E S E N T A**

Alexis Hernández Zamudio

**APROFESOR**

Jesús Hernández Cabrera

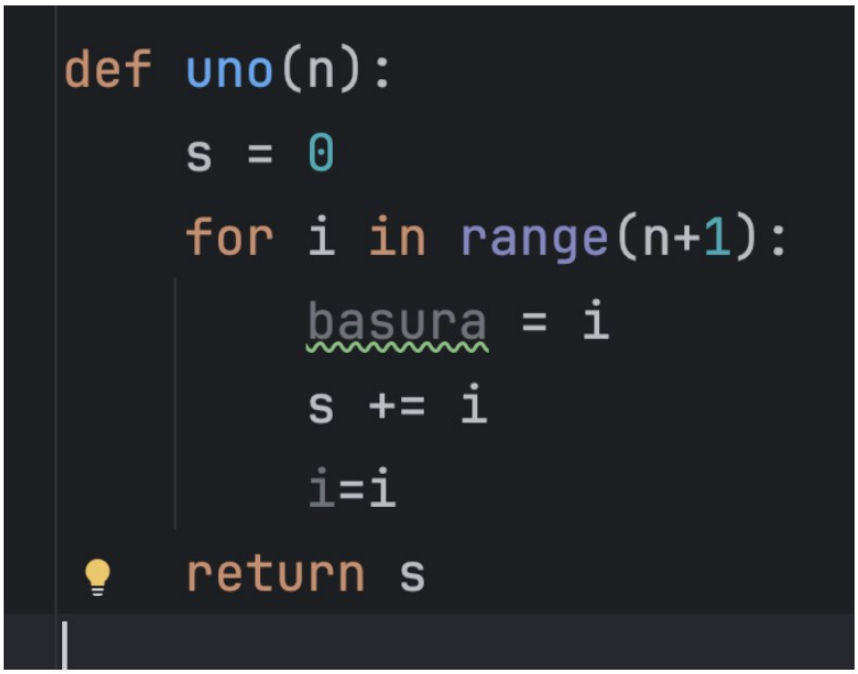
**Gpo:1558**

**URL del repositorio:**

**https://github.com/AlextyrB/Analisis_Algoritmos/tree/main**

**Ciudad Nezahualcóyotl, EDOMEX. 20 de Septiembre del 2025**

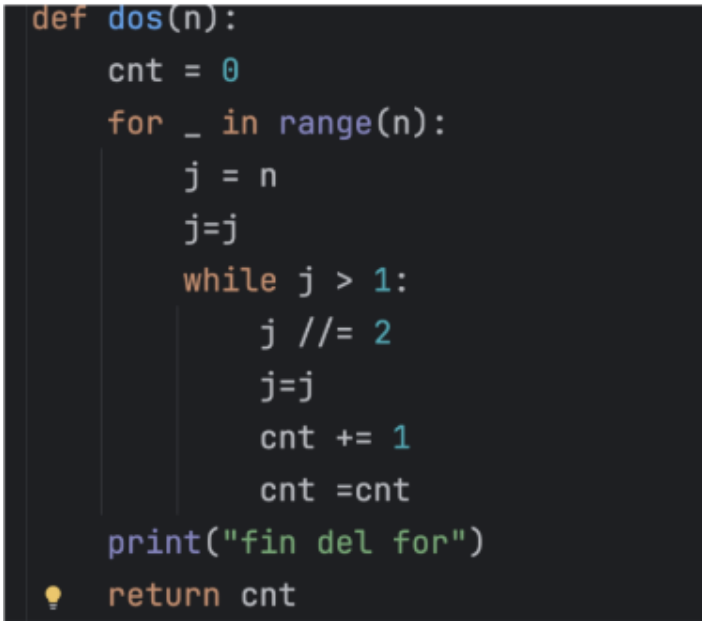
**Algoritmo 1**




**Análisis de Complejidad Temporal T(n):**
Inicialización: s = 0
Ejecuta 1 vez
Costo: T(n) = 1
Bucle for: for i in range(n+1)
Se ejecuta n+1 veces (de 0 a n inclusive)
Costo de evaluación del bucle: T(n) = n+1
Asignación basura: basura = i
Se ejecuta n+1 veces (una por cada iteración)
Costo: T(n) = n+1
Suma: s += i
Se ejecuta n+1 veces
Costo: T(n) = n+1
Reasignación innecesaria: i = i
Se ejecuta n+1 veces
Costo: T(n) = n+1
Return: return s
Ejecuta 1 vez
Costo: T(n) = 1
**Cálculo total T(n):**
**T(n) = 1 + (n+1) + (n+1) + (n+1) + (n+1) + 1 = 4(n+1) + 2 = 4n + 6**
En notación Big O: O(n) - lineal
**Análisis de Complejidad Espacial S(n):**
Parámetro: n
Ocupa 1 unidad de memoria
Costo: S(n) = 1

Variable s: s = 0
Ocupa 1 unidad de memoria
Costo: S(n) = 1
Variable del bucle: i in range(n+1)
La variable i ocupa 1 unidad de memoria
Costo: S(n) = 1
Variable basura: basura = i
Ocupa 1 unidad de memoria adicional
Costo: S(n) = 1
Operaciones restantes: s += i, i = i, return s
No requieren espacio adicional
Costo: S(n) = 0
Cálculo total S(n):
**S(n) = 1 + 1 + 1 + 1 + 0 = 4**
En notación Big O: O(1) - constante

**Algoritmo 2**

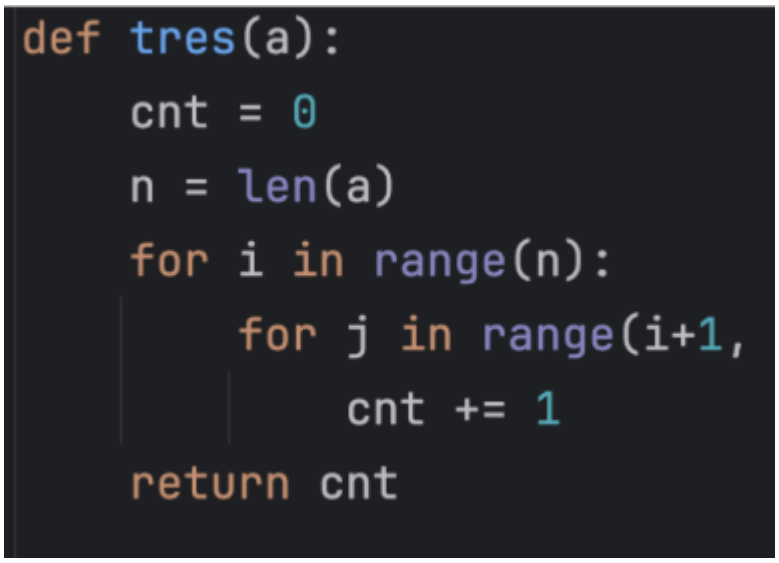


```
def dos(n):
    cnt = 0
    for _ in range(n):
        j = n
        j=j
        while j > 1:
            j //= 2
            j=j
            cnt += 1
            cnt =cnt
    print("fin del for")
    return cnt
```

**Análisis de Complejidad Temporal T(n):**
Inicialización: cnt = 0
Ejecuta 1 vez
Costo: T(n) = 1
Bucle for: for _ in range(n)
Se ejecuta n veces (de 0 a n-1)
Costo de evaluación del bucle: T(n) = n
Asignación j: j = n
Se ejecuta n veces (una por cada iteración del for)
Costo: T(n) = n
Reasignación innecesaria: j = j
Se ejecuta n veces
Costo: T(n) = n
Bucle while: while j > 1

Análisis crítico: Para cada iteración del for, j empieza en n y se divide por 2 hasta llegar a 1
Secuencia por iteración: j = n → n/2 → n/4 → n/8 → ... → 1
Número de iteraciones del while por cada iteración del for: $\log_2(n)$
Se ejecuta $n \times \log_2(n)$ veces en total
Costo de evaluación del while: $T(n) = n \times \log_2(n)$
División: j //= 2
Se ejecuta $n \times \log_2(n)$ veces
Costo: $T(n) = n \times \log_2(n)$
Reasignación innecesaria: j = j
Se ejecuta $n \times \log_2(n)$ veces
Costo: $T(n) = n \times \log_2(n)$
Incremento: cnt += 1
Se ejecuta $n \times \log_2(n)$ veces
Costo: $T(n) = n \times \log_2(n)$
Reasignación innecesaria: cnt = cnt
Se ejecuta $n \times \log_2(n)$ veces
Costo: $T(n) = n \times \log_2(n)$
Print: print("fin del for")
Ejecuta 1 vez
Costo: T(n) = 1
Return: return cnt
Ejecuta 1 vez
Costo: T(n) = 1
Cálculo total T(n):
$T(n) = 1 + n + n + n + n \times \log_2(n) + n \times \log_2(n) + n \times \log_2(n) + n \times \log_2(n) + n \times \log_2(n) + 1 + 1$
$T(n) = 3 + 3n + 5n \times \log_2(n)$
$T(n) = 5n \times \log_2(n) + 3n + 3$
**En notación Big O: O(n log n) – Logarítmica**
**Análisis de Complejidad Espacial S(n):**
Parámetro: n
Ocupa 1 unidad de memoria
Costo: S(n) = 1
Variable cnt: cnt = 0
Ocupa 1 unidad de memoria
Costo: S(n) = 1
Variable del bucle: _ in range(n)
La variable _ (aunque no se use) ocupa 1 unidad de memoria
Costo: S(n) = 1
Variable j: j = n
Ocupa 1 unidad de memoria
Costo: S(n) = 1
Operaciones restantes: j = j, while j > 1, j //= 2, cnt += 1, cnt = cnt, print(), return cnt
No requieren espacio adicional
Costo: S(n) = 0
Cálculo total S(n):
S(n) = 1 + 1 + 1 + 1 + 0 = 4
**En notación Big O: O(1) - constante**

**Algoritmo 3**



```
def tres(a):
    cnt = 0
    n = len(a)
    for i in range(n):
        for j in range(i+1,
            cnt += 1
    return cnt
```

**Análisis de Complejidad Temporal T(n):**
Inicialización: cnt = 0
Ejecuta 1 vez
Costo: T(n) = 1
Obtener longitud: n = len(a)
Ejecuta 1 vez
Costo: T(n) = 1
Bucle for externo: for i in range(n)
Se ejecuta n veces (de 0 a n-1)
Costo de evaluación del bucle: T(n) = n
Bucle for interno: for j in range(i+1, n)
Análisis crítico: Para cada valor de i, j va de (i+1) hasta (n-1)
Iteraciones por cada i:
i = 0: j va de 1 a n-1 → (n-1) iteraciones
i = 1: j va de 2 a n-1 → (n-2) iteraciones
i = 2: j va de 3 a n-1 → (n-3) iteraciones
...
i = n-2: j va de n-1 a n-1 → 1 iteración
i = n-1: j va de n a n-1 → 0 iteraciones
Total de iteraciones del bucle interno: (n-1) + (n-2) + (n-3) + ... + 1 + 0 = Σ(k=1 a n-1) k = (n-1)×n/2
Costo de evaluación del bucle interno: T(n) = (n-1)×n/2
Incremento: cnt += 1
Se ejecuta una vez por cada iteración del bucle interno
Se ejecuta (n-1)×n/2 veces en total
Costo: T(n) = (n-1)×n/2
Return: return cnt
Ejecuta 1 vez
Costo: T(n) = 1
Cálculo total T(n):

$T(n) = 1 + 1 + n + (n-1)\times n/2 + (n-1)\times n/2 + 1$
$T(n) = 3 + n + 2\times(n-1)\times n/2$
$T(n) = 3 + n + (n-1)\times n$
$T(n) = 3 + n + n^2 - n$
$T(n) = 3 + n^2$
$T(n) = n^2 + 3$
**En notación Big O: $O(n^2)$ - cuadrática**
**Análisis de Complejidad Espacial S(n):**
Parámetro: a (lista)
Variable cnt: cnt = 0
Ocupa 1 unidad de memoria
Costo: S(n) = 1
Variable n: n = len(a)
Ocupa 1 unidad de memoria
Costo: S(n) = 1
Variable del bucle externo: i in range(n)
La variable i ocupa 1 unidad de memoria
Costo: S(n) = 1
Variable del bucle interno: j in range(i+1, n)
La variable j ocupa 1 unidad de memoria
Costo: S(n) = 1
Operaciones restantes: cnt += 1, return cnt
No requieren espacio adicional
Costo: S(n) = 0
Cálculo total S(n):
S(n) = 1 + 1 + 1 + 1 + 0 = 4
**En notación Big O: O(1) - constante**

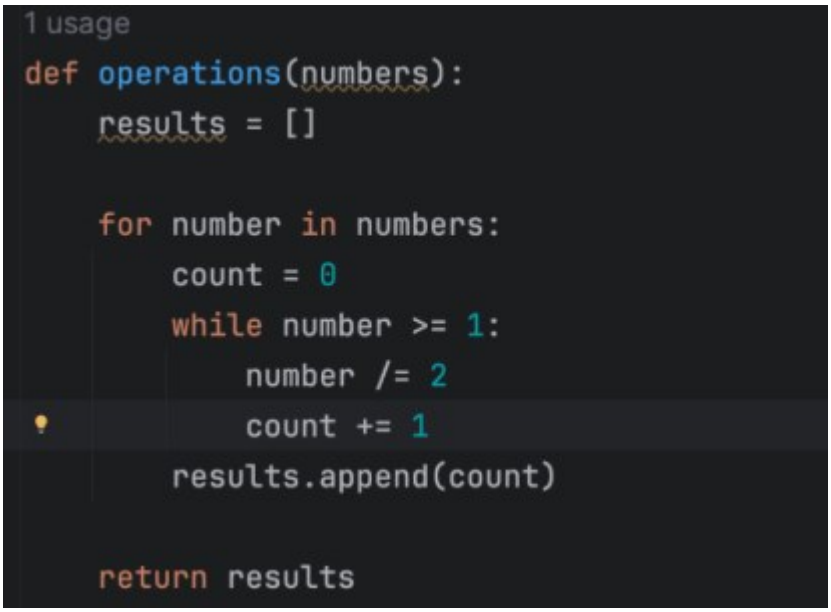
**Algoritmo 4**

```python
def cuatro(a):
    n = len(a)
    total = 0
    n= (n*2) / 2
    for i in range(n):
        basura = 1
        nada = 0
        i=i
        for j in range(n):
            basura2 = 1
            otra_cosa = 2
            j = j
            for k in range(n):
                total = total
                k=k
                j = j
                total += a[i] + a[j] + a[k]
    return total
```

**Análisis de Complejidad Temporal T(n):**
Obtener longitud: n = len(a)
Ejecuta 1 vez
Costo: T(n) = 1
Inicialización: total = 0
Ejecuta 1 vez
Costo: T(n) = 1
Operación redundante: n = (n*2) / 2
Ejecuta 1 vez (equivale a n = n, operación innecesaria)
Costo: T(n) = 1
Bucle for externo: for i in range(n)
Se ejecuta n veces (de 0 a n-1)
Costo de evaluación del bucle: T(n) = n
Asignaciones dentro del primer bucle: basura = 1, nada = 0, i = i
Se ejecutan n veces cada una (una por cada iteración del bucle i)
Costo: T(n) = 3n
Bucle for medio: for j in range(n)
Se ejecuta n veces por cada iteración de i
Total: $n \times n = n^2$ iteraciones
Costo de evaluación del bucle: $T(n) = n^2$
Asignaciones dentro del segundo bucle: basura2 = 1, otra_cosa = 2, j = j
Se ejecutan $n^2$ veces cada una
Costo: $T(n) = 3n^2$
Bucle for interno: for k in range(n)
Se ejecuta n veces por cada iteración de j, que a su vez se ejecuta n veces por cada iteración de i
Total: $n \times n \times n = n^3$ iteraciones
Costo de evaluación del bucle: $T(n) = n^3$
Asignaciones dentro del tercer bucle: total = total, k = k, j = j
Se ejecutan $n^3$ veces cada una
Costo: $T(n) = 3n^3$
Operación principal: total += a[i] + a[j] + a[k]
Se ejecuta $n^3$ veces
Incluye:
3 accesos a la lista: $3n^3$
2 sumas: $2n^3$
1 asignación: $n^3$
Costo: $T(n) = 6n^3$
Return: return total
Ejecuta 1 vez
Costo: T(n) = 1
Cálculo total T(n):
$T(n) = 1 + 1 + 1 + n + 3n + n^2 + 3n^2 + n^3 + 3n^3 + 6n^3 + 1$
$T(n) = 4 + 4n + 4n^2 + 10n^3$
$T(n) = 10n^3 + 4n^2 + 4n + 4$
**En notación Big O: $O(n^3)$ - cúbica**
**Análisis de Complejidad Espacial S(n):**
Parámetro: a (lista)
La lista a ocupa n unidades de memoria (parámetro de entrada)
Para el análisis de espacio auxiliar: S(n) = 0 (no cuenta como espacio adicional)
Variable n: n = len(a)

Ocupa 1 unidad de memoria
Costo: S(n) = 1
Variable total: total = 0
Ocupa 1 unidad de memoria
Costo: S(n) = 1
Variable del primer bucle: i in range(n)
La variable i ocupa 1 unidad de memoria
Costo: S(n) = 1
Variables dentro del primer bucle: basura, nada
Ocupan 2 unidades de memoria
Costo: S(n) = 2
Variable del segundo bucle: j in range(n)
La variable j ocupa 1 unidad de memoria
Costo: S(n) = 1
Variables dentro del segundo bucle: basura2, otra_cosa
Ocupan 2 unidades de memoria
Costo: S(n) = 2
Variable del tercer bucle: k in range(n)
La variable k ocupa 1 unidad de memoria
Costo: S(n) = 1
Operaciones restantes: reasignaciones y accesos a lista
No requieren espacio adicional
Costo: S(n) = 0
Cálculo total S(n):
S(n) = 1 + 1 + 1 + 2 + 1 + 2 + 1 + 0 = 9
**En notación Big O: O(1) - constante**

**Algoritmo 5**



**Análisis de Complejidad Temporal T(n):**
Inicialización: results = []
Ejecuta 1 vez
Costo: T(n) = 1
Bucle for externo: for number in numbers
Se ejecuta n veces (donde n = len(numbers))
Costo de evaluación del bucle: T(n) = n
Inicialización del contador: count = 0
Se ejecuta n veces (una por cada iteración del for)
Costo: T(n) = n
Bucle while interno: while number >= 1
Análisis crítico: Para cada número en la lista, se divide por 2 hasta ser menor que 1
Si el número inicial es M, la secuencia es: M → M/2 → M/4 → M/8 → ... → M/2^k < 1
Número de iteraciones por número: $\lfloor \log_2(M) \rfloor + 1$
Caso importante: La complejidad depende de los valores en la lista, no solo de la longitud
División: number /= 2
Se ejecuta $\sum \log_2(M_i)$ veces para todos los números $M_i$ en la lista
Costo: T(n) = $\sum \log_2(M_i)$
Incremento: count += 1
Se ejecuta $\sum \log_2(M_i)$ veces
Costo: T(n) = $\sum \log_2(M_i)$
Append: results.append(count)
Se ejecuta n veces
Costo: T(n) = n
Return: return results
Ejecuta 1 vez
Costo: T(n) = 1
Cálculo total T(n):
T(n) = 1 + n + n + $\sum \log_2(M_i)$ + $\sum \log_2(M_i)$ + n + 1
T(n) = 3 + 3n + 2$\sum \log_2(M_i)$
Casos de análisis:
Mejor caso: Todos los números son muy pequeños (≤ 1) → O(n)
Caso promedio: Números de tamaño típico → O(n × log(M_promedio))
Peor caso: Números muy grandes → O(n × log(M_max))
**En notación Big O (caso general): O(n × log(M)) donde M es el valor típico de los números**

**Análisis de Complejidad Espacial S(n):**
Parámetro: numbers (lista)
Para el análisis de espacio auxiliar: S(n) = 0 (no cuenta como espacio adicional)
Lista results: results = []
Crece hasta tener n elementos (uno por cada número procesado)
Costo: S(n) = n
Variable del bucle externo: number in numbers
La variable number ocupa 1 unidad de memoria
Costo: S(n) = 1
Variable count: count = 0
Ocupa 1 unidad de memoria
Costo: S(n) = 1

Operaciones restantes: divisiones, incrementos, append, return
No requieren espacio adicional permanente
Costo: S(n) = 0
Cálculo total S(n):
S(n) = n + 1 + 1 + 0 = n + 2
**En notación Big O: O(n) - lineal**