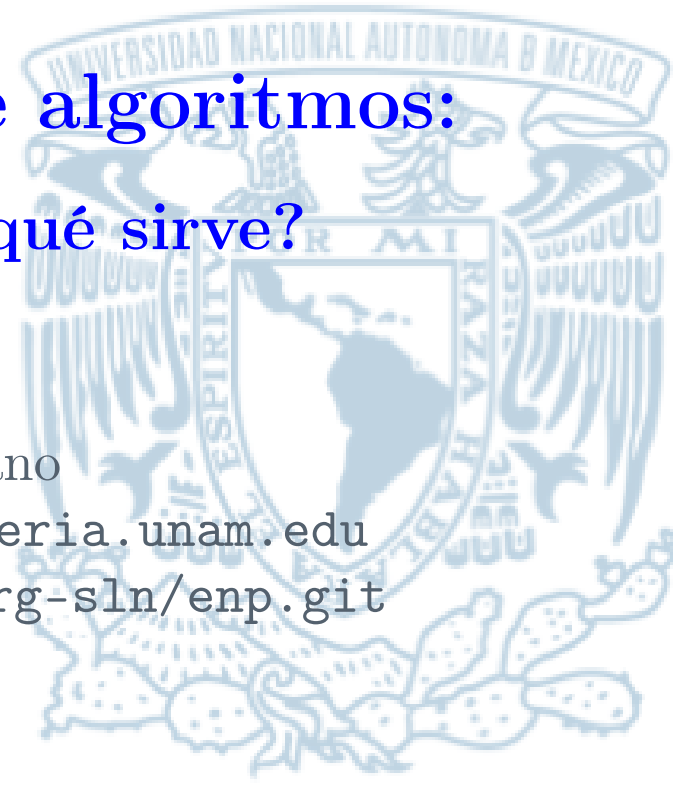
Departamento de Computación
DIE, FI, UNAM



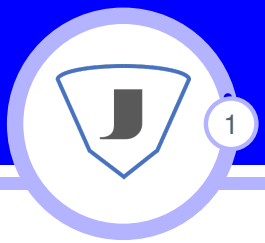
Análisis básico de algoritmos:

¿qué es y para qué sirve?

Jorge Solano
jorge.solano@ingenieria.unam.edu
https://github.com/jrg-sln/enp.git

Noviembre 19, 2021

Content



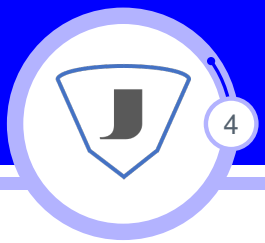
Análisis de algoritmos

El análisis de algoritmos permite conocer la eficiencia de una pieza de código.

Centro de Comunicación de las Ciencias de la Universidad Autónoma - Creative Commons Attribution-Share Alike 4.0

¿En función de qué se mide la eficiencia de un algoritmo?

Medidas de eficiencia
Análisis de algoritmos



- Ancho de banda (uso de red).
- Localidades de memoria.
- Ciclos de reloj del procesador.

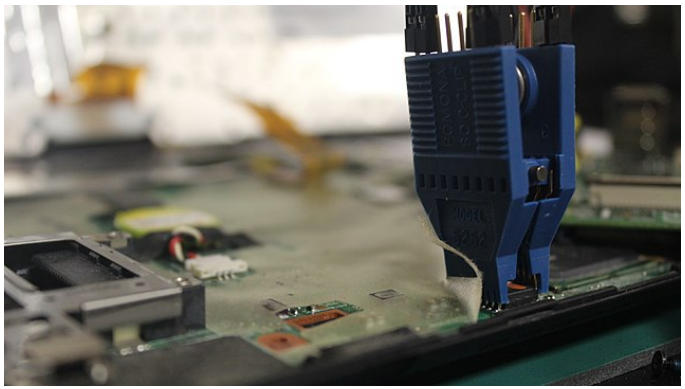
¿Y cómo se mide la eficiencia de un algoritmo de forma teórica?

Medidas de eficiencia
Análisis de algoritmos

Las dos técnicas más utilizadas para medir la eficiencia de un algoritmo sin necesidad de implementarlo son:

- Modelo RAM
- Análisis asintótico

El modelo RAM es una representación de una computadora hipotética que permite evaluar la eficiencia de un algoritmo de manera independiente del hardware de la computadora.



Magurale - Creative Commons Attribution-Share Alike 4.0

El análisis bajo el modelo RAM sigue las siguientes reglas:

- Las operaciones simples (+, -, *, /, selección o llamada a función) toman exactamente un ciclo de reloj.

El análisis bajo el modelo RAM sigue las siguientes reglas:

- Las operaciones simples (+, -, *, /, selección o llamada a función) toman exactamente un ciclo de reloj.
- Los ciclos están compuestos por operaciones simples, por lo tanto, el tiempo que toma un ciclo depende del número de iteraciones del mismo.

El análisis bajo el modelo RAM sigue las siguientes reglas:

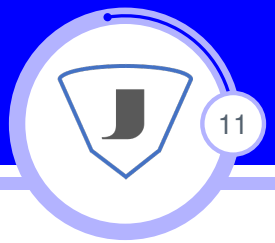
- Las operaciones simples (+, -, *, /, selección o llamada a función) toman exactamente un ciclo de reloj.
- Los ciclos están compuestos por operaciones simples, por lo tanto, el tiempo que toma un ciclo depende del número de iteraciones del mismo.
- El acceso a memoria toma exactamente un paso de tiempo. La memoria de un modelo RAM es infinita.

Determinar si un número es par o impar

Requerimientos

Dado un número entero n determinar si el número es par o impar.

- Si n es par, regresar 0.
- Si n es impar, regresar 1.



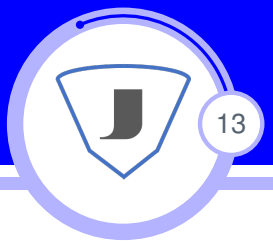
Análisis

Entrada: un número entero n.

Salida: un número entero *bit*, si n es par $bit = 0$, si n es impar $bit = 1$.

Diseño

```
FUNCTION odd_even (INTEGER: n) RETURN INTEGER:
    INTEGER bit
    if (n % 2 == 0)
        bit = 0
    else
        bit = 1
    return bit
END FUNCTION
```



Diseño (análisis teórico)

```
FUNCTION odd_even (INTEGER: n) RETURN INTEGER:
    INTEGER bit          # c1
    if (n % 2 == 0)      # c2
        bit = 0          # c3
    else
        bit = 1          # c4
    return bit           # c5
END FUNCTION
```

Diseño (análisis teórico)

```
FUNCTION odd_even (INTEGER: n) RETURN INTEGER:
    INTEGER bit          # c1
    if (n % 2 == 0)      # c2
        bit = 0          # c3
    else
        bit = 1          # c4
    return bit           # c5
END FUNCTION
```

$$f(n) = c1 + c2 + c3 + c4 + c5$$

Diseño (análisis teórico)

```
FUNCTION odd_even (INTEGER: n) RETURN INTEGER:
    INTEGER bit          # c1
    if (n % 2 == 0)      # c2
        bit = 0          # c3
    else
        bit = 1          # c4
    return bit           # c5
END FUNCTION
```

$$f(n) = c1 + c2 + c3 + c4 + c5$$
$$f(n) = c$$

Diseño (análisis teórico)

```
FUNCTION odd_even (INTEGER: n) RETURN INTEGER:
    INTEGER bit          # c1
    if (n % 2 == 0)      # c2
        bit = 0          # c3
    else
        bit = 1          # c4
    return bit           # c5
END FUNCTION
```

$$f(n) = c1 + c2 + c3 + c4 + c5$$
$$f(n) = c$$
$$f(n) = O(1)^1$$

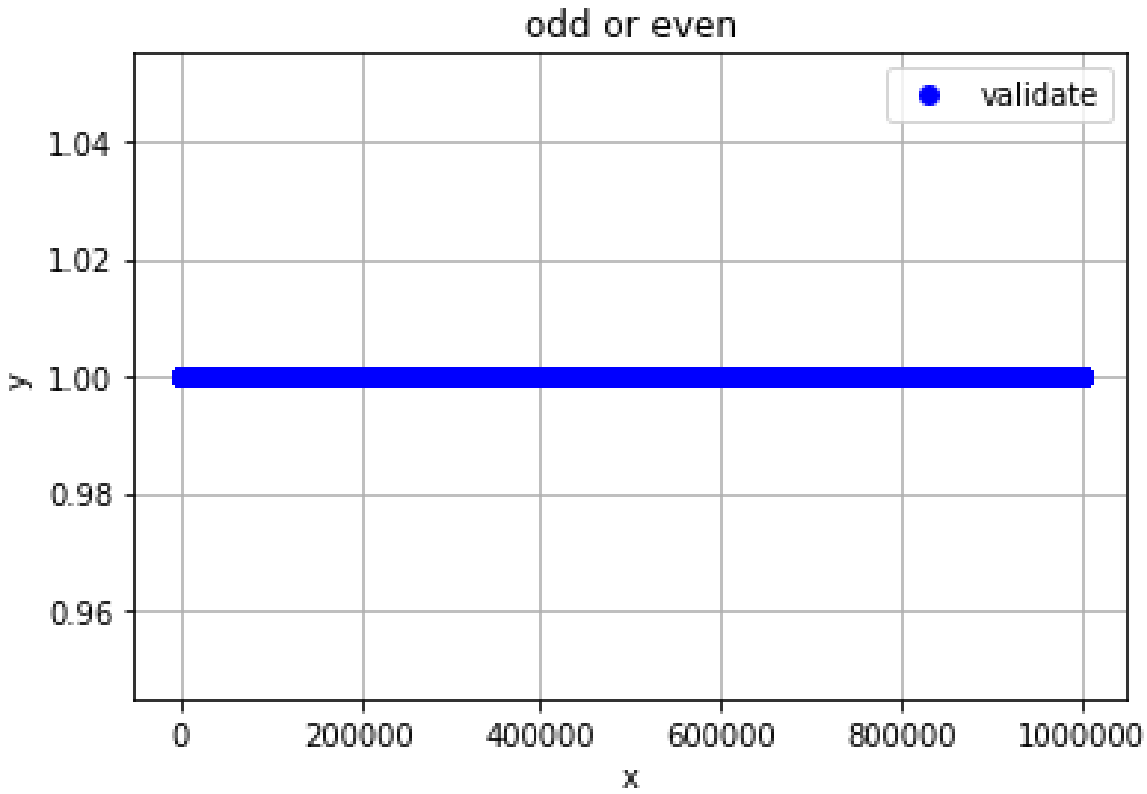
[1] Esta notación se conoce como notación O mayúsucla, notación Omicron o notación Big-O

Implementación

```
def odd_even(intN):
    bit = 0
    if intN%2 == 0:
        bit = 0
    else:
        bit = 1

    return bit
```

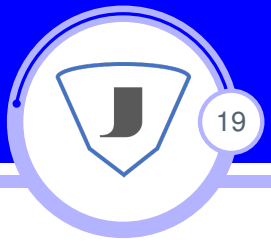
Implementación (análisis práctico)



Convertir un número entero de base 10 a base 2

Requerimientos

Dado un número entero *n* en base 10, transformarlo a binario.



Análisis

Entrada: un número entero *n* en base 10.

Salida: cadena binaria del número *n*.

Diseño

```
FUNCTION dec2bin (INTEGER: n) RETURN STRING:
    bin = ''
    while (n>0):
        bin.insert(0, n%2)
        n = n // 2
    return bin
END FUNCTION
```

Diseño (análisis teórico)

```
FUNCTION dec2bin (INTEGER: n) RETURN STRING:
    bin = ''                        # c1
    while (n>0):                    # c2 * (log(n)+1)
        bin.insert(0, n%2)          # c3 * log(n)
        n = n // 2                  # c4 * log(n)
    return bin                      # c5
END FUNCTION
```

Diseño (análisis teórico)

```
FUNCTION dec2bin (INTEGER: n) RETURN STRING:
    bin = ''                        # c1
    while (n>0):                    # c2 * (log(n)+1)
        bin.insert(0, n%2)          # c3 * log(n)
        n = n // 2                  # c4 * log(n)
    return bin                      # c5
END FUNCTION
```

$$f(n) = (c2+c3+c4)*\log(n) + (c1+c2+c5)$$

Diseño (análisis teórico)

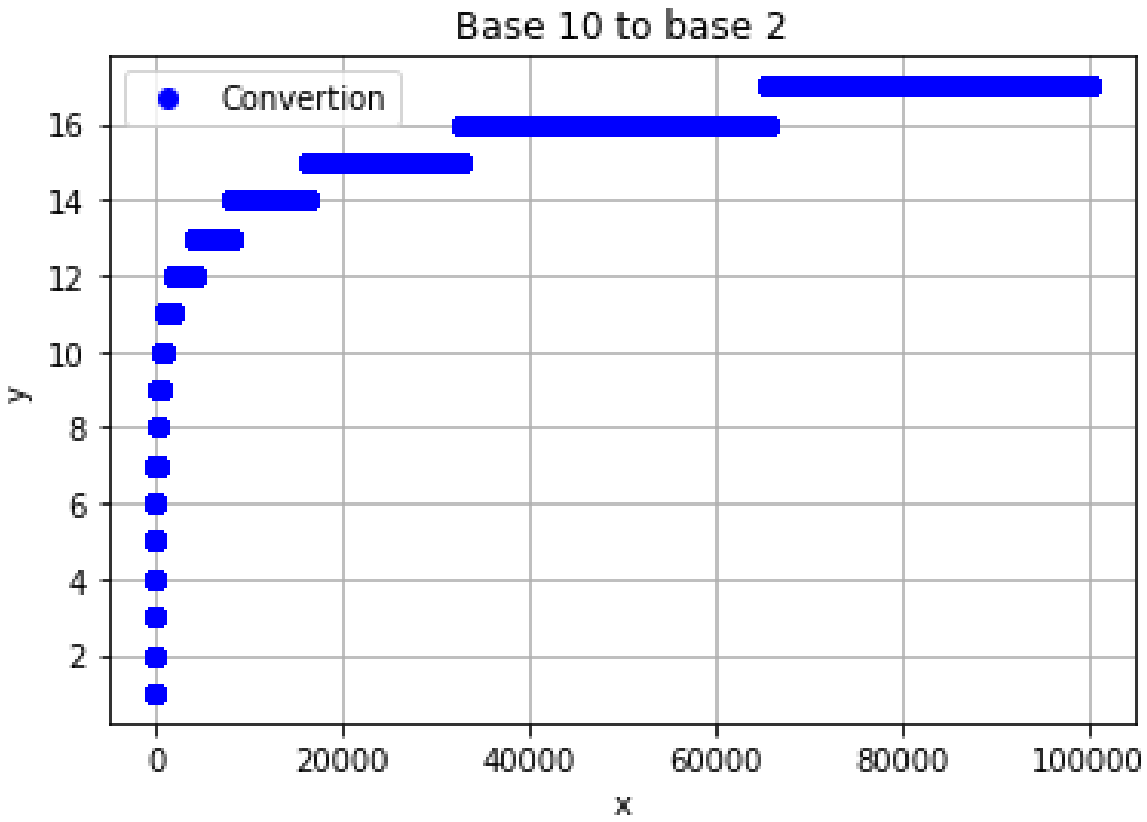
```
FUNCTION dec2bin (INTEGER: n) RETURN STRING:
    bin = ''                        # c1
    while (n>0):                    # c2 * (log(n)+1)
        bin.insert(0, n%2)          # c3 * log(n)
        n = n // 2                  # c4 * log(n)
    return bin                      # c5
END FUNCTION
```

$$f(n) = (c2+c3+c4)*\log(n) + (c1+c2+c5)$$
$$f(n) = O(\log(n))$$

Implementación

```
def dec2bin(intN):
    bin = ''
    while intN>0:
        bin = bin + str(intN%2)
        intN //= 2
    return bin
```

Implementación (análisis práctico)



Suma de los valores de una lista

Requerimientos

Dado un conjunto c de n elementos enteros $c = [e_1, \cdots, e_n]$, obtener la suma de todos los elementos, es decir, $suma = e_1 + \cdots + e_n$.

Análisis

Entrada: un conjunto *c* de *n* elementos enteros $[e_1, \cdots, e_n]$.

Salida: un número entero *suma* que contenga la suma de todos los *n* elementos del conjunto *c*.

Diseño

```
FUNCTION sumar_conjunto (INTEGER: c[]) RETURN INTEGER:
    INTEGER i = 0, suma = 0
    while i < len(c):
        suma = suma + c[i]
        i = i+1
    return suma
END FUNCTION
```

Diseño (análisis teórico)

```
FUNCTION sumar_conjunto (INTEGER: c[]) RETURN INTEGER:
    INTEGER i = 0, suma = 0      # c1
    while i < len(c):            # c2 * (n + 1)
        suma = suma + c[i]       # c3 * n
        i = i+1                  # c4 * n
    return suma                  # c5
END FUNCTION
```

Diseño (análisis teórico)

```
FUNCTION sumar_conjunto (INTEGER: c[]) RETURN INTEGER:
    INTEGER i = 0, suma = 0      # c1
    while i < len(c):            # c2 * (n + 1)
        suma = suma + c[i]       # c3 * n
        i = i+1                  # c4 * n
    return suma                  # c5
END FUNCTION
```

$$f(n) = (c2+c3+c4)*n + (c1+c2+c5)$$

Diseño (análisis teórico)

```
FUNCTION sumar_conjunto (INTEGER: c[]) RETURN INTEGER:
    INTEGER i = 0, suma = 0      # c1
    while i < len(c):            # c2 * (n + 1)
        suma = suma + c[i]       # c3 * n
        i = i+1                  # c4 * n
    return suma                  # c5
END FUNCTION
```

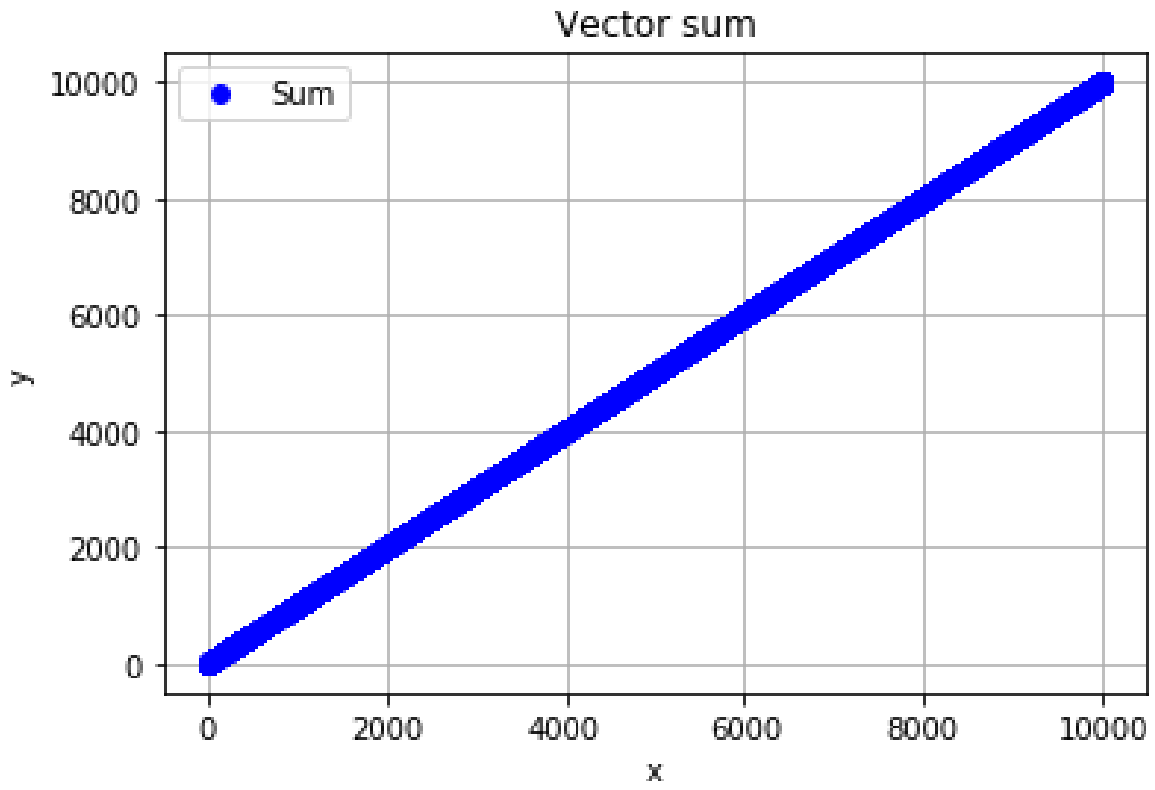
$$f(n) = (c2+c3+c4)*n + (c1+c2+c5)$$
$$f(n) = O(n)$$

Implementación

```
def vector_sum(conjunto):
    i = 0
    suma = 0
    while i < len(conjunto):
        suma += conjunto[i]
        i += 1

    return suma
```

Implementación (análisis práctico)



Resumen

El análisis de algoritmos permite:

- Predecir los recursos computacionales que un algoritmo va a ocupar durante su ejecución.
- Comparar eficiencia entre algoritmos.
- Determinar si un algoritmo es viable de implementarse o no.

¡Muchas gracias!

jorge.solano@ingenieria.unam.edu

https://github.com/jrg-sln/enp.git