Trabajo practico integrador: Análisis de Algoritmos en Python

Titulo: Análisis de algoritmos

Alumnos:

Brian Santaran (brian.santaran@tupad.utn.edu.ar)

Michel Emmanuel San Martin (mikeemmanuel91@gmail.com)

Introducción

El análisis de algoritmos ha sido elegido como tema a desarrollar en el presente trabajo dado que resulta de un pilar fundamental para la optimización y crecimiento de la industria del software.

En la programación es de vital importancia dado que permite analizar la eficiencia de un algoritmo, en términos de pasos a ejecutar, consumo de recursos disponibles y permitir trazar su escalabilidad, dado que estas pruebas teóricas se plantean utilizando N cantidad de datos, a modo de determinar la eficiencia con un volumen de datos cada vez más grande.

El presente trabajo busca exponer las formas de análisis tanto teórico como empírico para determinar la eficiencia de un algoritmo desarrollado en Python.

Marco teórico

El análisis de algoritmos se sirve de dos enfoques:

- Análisis teórico: Enfoque matematico que busca calcular una función temporal T(n) donde se representa el número de operaciones del algoritmo acorde a un ‘n’ tamaño de entrada. Enfoque independiente del hardware o software.
- Análisis empírico: Enfoque de pruebas donde se realiza la ejecución del algoritmo y se mide su tiempo de ejecución, sirviéndose de módulos como time() en Python. Este enfoque depende del hardware y software que se utilice, motivo por el cual para tener consistencia en las pruebas es mandatorio realizarlas en un mismo ambiente.

Caso practico

Se plantean dos algoritmos que recibirán por parámetros una lista/array y un string (en este caso puntual ingresados por el usuario).

Objetivo: Recibir un nombre de pokemon y buscar si se encuentra o no dentro de una lista predeterminada; retornando al usuario un mensaje en caso afirmativo y el número de índice, en caso de no haber coincidencia se devuelve un mensaje informando al usuario.
(*) Como ejercicio inicial se plantea una lista de 150 elementos (pokemones).

(**) Como ejercicio adicional se incluye una lista de 1025 elementos (Totalidad catalogada de pokemones)

Como paso previo a la definición de algoritmos, se define función de tiempo para manejo de pruebas empíricas:

```
import time
def use_time():
used_time = time.time()
return used_time

def time_dif(start_time, end_time):
total_time = end_time - start_time
print(f"El tiempo transcurrido en la ejecucion fue de: {total_time:.8f} segundos.")
```

Se definen algoritmos de búsqueda:

```
# Algoritmo_1
def buscar_combinacion_bruto(lista, objetivo):
start = use_time()
print("Comienzo de Algoritmo_1.")
objetivo = objetivo.upper()

for nombre in lista:
if nombre.upper() == objetivo:
end=use_time()
time_dif(start, end)
print(f"El Pokemon ingresado es el numero #{lista.index(nombre) + 1}")
return True
else:
end = use_time()
time_dif(start, end)
print("El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.")
return False
```

```
# Algoritmo_2
def busqueda_en_diccionario(lista, objetivo):
start = use_time()
print("Comienzo de Algoritmo_2")

objetivo = objetivo.upper()
poke_index = {nombre.upper():i + 1 for i, nombre in enumerate(lista)}

if objetivo in poke_index:
end = use_time()
time_dif(start, end)
print(f"El pokemon ingresado es el numero #{poke_index[objetivo]}")
return True
```

```
else:
end = use_time()
time_dif(start, end)
print("El nombre del pokemon ingresado no es valido o este no pertenece a la region de Kanto.")
return False
```

Análisis empírico:

Ejecucion del programa:
(*) Se ingresa el valor “Mewtwo” que coincide con el ultimo valor del array.

Por favor, ingrese el nombre de un pokemon: Mewtwo

Comienzo de Algoritmo_1.

El tiempo transcurrido en la ejecucion fue de: 0.00004244 segundos.

El Pokemon ingresado es el numero #150

Comienzo de Algoritmo_2

El tiempo transcurrido en la ejecucion fue de: 0.00002766 segundos.

El pokemon ingresado es el numero #150

Resultados:
Algoritmo_1 tiempo de ejecucion: 0.00004244 segundos.

Algoritmo_2 tiempo de ejecucion: 0.00002766 segundos.

Ejecucion del programa:

(*) Segunda ejecucion, se ingresa el valor “Agumon” que no se encuentra la lista.

Por favor, ingrese el nombre de un pokemon: Agumon

Comienzo de Algoritmo_1.

El tiempo transcurrido en la ejecucion fue de: 0.00004387 segundos.

El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.

Comienzo de Algoritmo_2

El tiempo transcurrido en la ejecucion fue de: 0.00003242 segundos.

El nombre del pokemon ingresado no es valido o este no pertenece a la region de Kanto.

Resultados:

Algoritmo_1 tiempo de ejecucion: 0.00004387 segundos.

Algoritmo_2 tiempo de ejecucion: 0.00003242 segundos.

Ejecucion del programa:
(**) Se ingresa el valor “Pecharunt” que coincide con el ultimo valor del array.

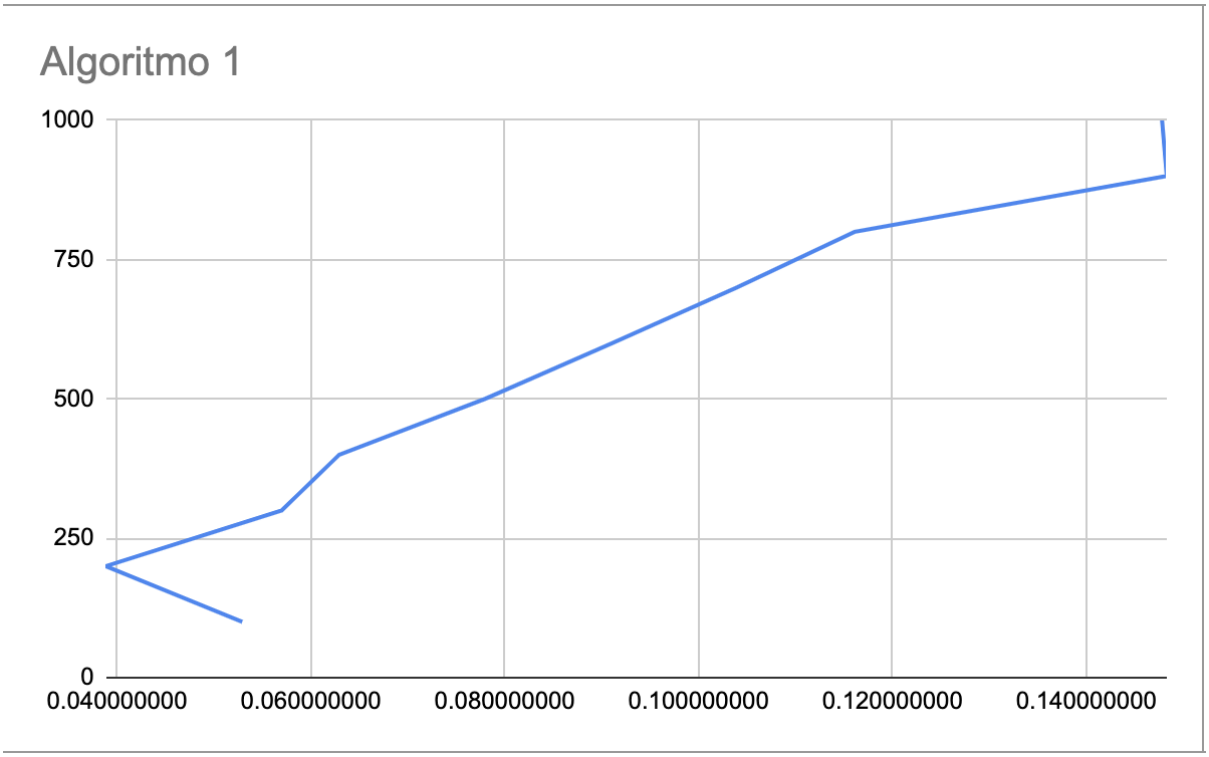
```
→  UTN-TUPaD-P1-TPI git:(main) ✗ python main.py
Por favor, ingrese el nombre de un pokemon: Pecharunt
Pruebas empíricas con las listas de Pokémon: 100
Comienzo de Algoritmo_1.
El tiempo transcurrido en la ejecucion fue de: 0.00005293 segundos.
El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 200
Comienzo de Algoritmo_1.
El tiempo transcurrido en la ejecucion fue de: 0.00003886 segundos.
El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 300
Comienzo de Algoritmo_1.
El tiempo transcurrido en la ejecucion fue de: 0.00005698 segundos.
El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 400
Comienzo de Algoritmo_1.
El tiempo transcurrido en la ejecucion fue de: 0.00006294 segundos.
El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 500
Comienzo de Algoritmo_1.
El tiempo transcurrido en la ejecucion fue de: 0.00007796 segundos.
El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 600
Comienzo de Algoritmo_1.
El tiempo transcurrido en la ejecucion fue de: 0.00009108 segundos.
El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 700
Comienzo de Algoritmo_1.
El tiempo transcurrido en la ejecucion fue de: 0.00010395 segundos.
El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 800
Comienzo de Algoritmo_1.
El tiempo transcurrido en la ejecucion fue de: 0.00011611 segundos.
El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 900
Comienzo de Algoritmo_1.
El tiempo transcurrido en la ejecucion fue de: 0.00014830 segundos.
El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 1000
Comienzo de Algoritmo_1.
El tiempo transcurrido en la ejecucion fue de: 0.00014782 segundos.
El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.
```

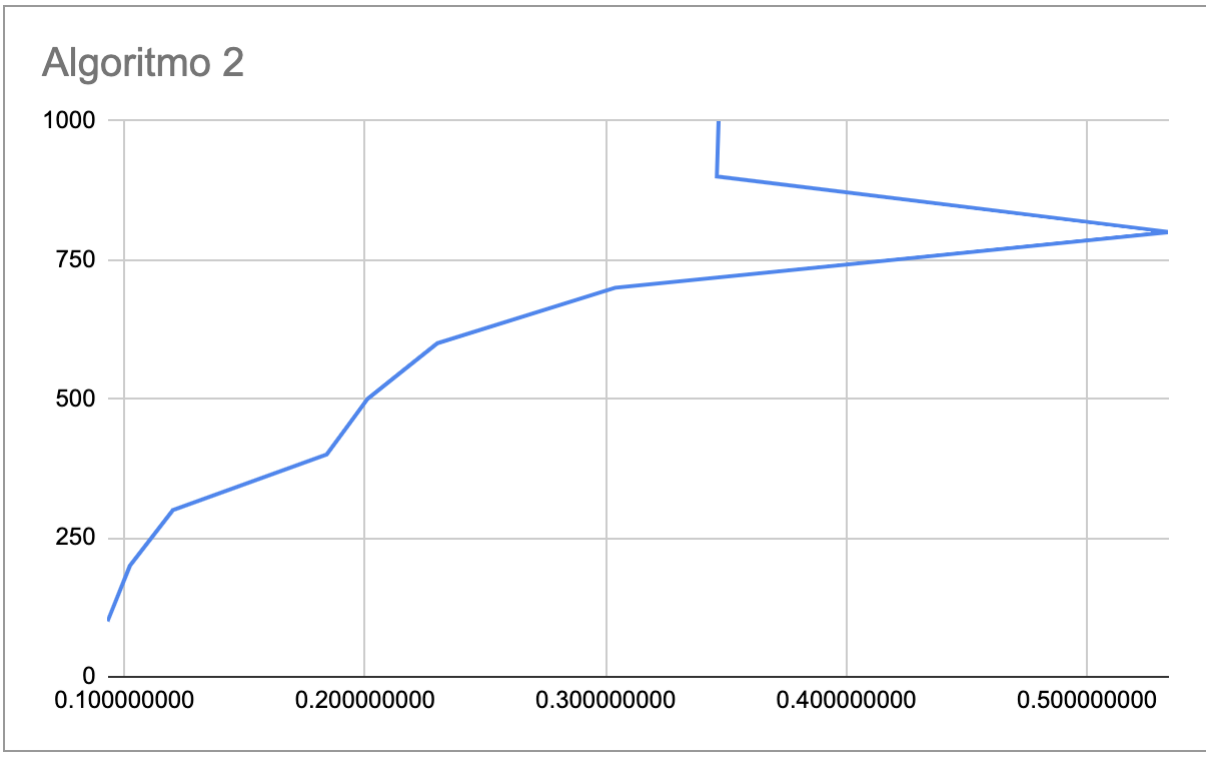
```
Pruebas empíricas con las listas de Pokémon usando diccionario:
Pruebas empíricas con las listas de Pokémon: 100
Comienzo de Algoritmo_2
El tiempo transcurrido en la ejecucion fue de: 0.00009298 segundos.
El nombre del pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 200
Comienzo de Algoritmo_2
El tiempo transcurrido en la ejecucion fue de: 0.00010228 segundos.
El nombre del pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 300
Comienzo de Algoritmo_2
El tiempo transcurrido en la ejecucion fue de: 0.00012016 segundos.
El nombre del pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 400
Comienzo de Algoritmo_2
El tiempo transcurrido en la ejecucion fue de: 0.00018406 segundos.
El nombre del pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 500
Comienzo de Algoritmo_2
El tiempo transcurrido en la ejecucion fue de: 0.00020099 segundos.
El nombre del pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 600
Comienzo de Algoritmo_2
El tiempo transcurrido en la ejecucion fue de: 0.00023007 segundos.
El nombre del pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 700
Comienzo de Algoritmo_2
El tiempo transcurrido en la ejecucion fue de: 0.00030398 segundos.
El nombre del pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 800
Comienzo de Algoritmo_2
El tiempo transcurrido en la ejecucion fue de: 0.00053406 segundos.
El nombre del pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 900
Comienzo de Algoritmo_2
El tiempo transcurrido en la ejecucion fue de: 0.00034618 segundos.
El nombre del pokemon ingresado no es valido o este no pertenece a la region de Kanto.
Pruebas empíricas con las listas de Pokémon: 1000
Comienzo de Algoritmo_2
El tiempo transcurrido en la ejecucion fue de: 0.00034690 segundos.
El pokemon ingresado es el numero #1025
```

Resultados:

Cantidad	Algoritmo 1	Milisegundos	Algoritmo 2	Milisegundos
100	0.000052930	0.052930000	0.000092980	0.092980000
200	0.000038860	0.038860000	0.000102280	0.102280000
300	0.000056980	0.056980000	0.000120160	0.120160000
400	0.000062940	0.062940000	0.000184060	0.184060000
500	0.000077960	0.077960000	0.000200990	0.200990000
600	0.000091080	0.091080000	0.000230070	0.230070000
700	0.000103950	0.103950000	0.000303980	0.303980000
800	0.000116110	0.116110000	0.000534060	0.534060000
900	0.000148300	0.148300000	0.000346180	0.346180000
1000	0.000147820	0.147820000	0.000346900	0.346900000

Graficos


Algoritmo 1
1000
750
500
250
0
0.040000000
0.060000000
0.080000000
0.100000000
0.120000000
0.140000000


Algoritmo 2
1000
750
500
250
0
0.100000000
0.200000000
0.300000000
0.400000000
0.500000000

Análisis teórico:

buscar_combinacion_bruto

```python
# Algoritmo_1
def buscar_combinacion_bruto(lista, objetivo):
    start = use_time()
    print("Comienzo de Algoritmo_1.") ## 1
    objetivo = objetivo.upper() ## 2

    for nombre in lista: ## n
        if nombre.upper() == objetivo: ## 2
            end=use_time()
            time_dif(start, end)
            print(f"El Pokemon ingresado es el numero #{lista.index(nombre) + 1}") ## 2
            return True ## 1

    else:
        end = use_time()
        time_dif(start, end)
        print("El nombre de pokemon ingresado no es valido o este no pertenece a la region de Kanto.") ## 1
        return False ## 1
```

Mejor caso (el Pokémon está primero en la lista):

- 1 (print)
- 2 (upper + asignacion)
- 2*n (upper + comparacion)
- n + 2 (index, print, suma)
- **1 (return)**
- **T(n) = 1 + 2 + n + 2 + 2 = n + 7**

Peor caso (esta ultimo):

- 1 (upper)
- 2 (upper + asignacion)
- 2n (upper + comparacion)
- 2 + n (indexado + suma + print)
- 1 (return)
- **T(n) = 1 + 2 + 2n + 2 + n + 1 = 3n + 6**

Conclusión

T(n) ≈ 3n + c→ O(n)

El algoritmo es **lineal** en complejidad temporal, ya que compara secuencialmente hasta encontrar coincidencia o terminar la lista.

busqueda_en_diccionario

```python
def busqueda_en_diccionario(lista, objetivo):
    start = use_time()
    print("Comienzo de Algoritmo_2") ## 1

    objetivo = objetivo.upper() ## 1
    poke_index = {}

    for i, nombre in enumerate(lista): ## n
        poke_index[nombre.upper()] = i + 1 ## 3

    if objetivo in poke_index: ## 1
        end = use_time()
        time_dif(start, end)
        print(f"El pokemon ingresado es el numero #{poke_index[objetivo]}") ## 2
        return True ## 1
    else:
        end = use_time()
        time_dif(start, end)
        print("El nombre del pokemon ingresado no es valido o este no pertenece a la region de Kanto.") ## 1
        return False ## 1
```

Mejor caso (el Pokémon está primero en la lista):

- 1 (print)
- 2 (upper + asignacion)
- 1 (asignacion)
- 3n (upper + suma + asignacion)
- 3 (index + print + ingreso al diccionario)
- **1 (return)**
- **T(n) = 1 + 2 + 1 + 3n + 3 +1 = 3n + 8**

Peor caso (esta ultimo):

- 1 (print)
- 2 (upper + asignacion)
- 1 (asignacion)
- 3n (upper + suma + asignacion)
- 3 (index + print + ingreso al diccionario)
- **1 (return)**

- **T(n) = 1 + 2 + 1 + 3n + 3 +1 = 3n + 8**

Conclusión

T(n) ≈ 3n + c→ O(n)

El algoritmo es **lineal** en complejidad temporal, ya que compara secuencialmente hasta encontrar coincidencia o terminar la lista.

El mejor y peor caso, son iguales ya que en ambos se recorre toda la lista para generar el diccionario y luego se accede al valor en caso exista la key.

Resultados obtenidos y conclusiones

- Ambos algoritmos resuelven el problema de busqueda planteado.
- A niveles teoricos, ambos son simililares ya que ambos son O(n)
- A niveles prácticos, se comportan de manera similar, teniendo como peor caso, el recorrido completo del listado para ambos algoritmos
- La diferencia se da en casos donde el pokemon buscado se encuentra en posición diferente al final, ya que el algoritmo 1, corta su ejecución al encontrarlo, mientras que el algoritmo 2, recorre todo el listado para formar el diccionario. (tener en cuenta que esta diferencia es muy poca en tiempo)

Metodología utilizada

- Planteo de problema de búsqueda y variantes en el enfoque del mismo.
- Codificacion en Python, prueba, error y ajuste del código.
- Ajustes de código mediante commits en GIT.
- Medición de resultados mediante la función time().
- Documentación en GIT.

Bibliografía

- Documentación oficial Python: https://docs.python.org/
- Wikidex Base de datos Pokémon https://www.wikidex.net/wiki/Lista_de_Pok%C3%A9mon
- Notación Big O | Análisis de algoritmos de forma sencilla: Youtube
- UTN – FRBA. (2025). *Apuntes de Programación I: Análisis Teórico y Empírico*.
- Guía: Notación Big O - Gráfico de complejidad de tiempo: **https://www.freecodecamp.org/espanol/news/hoja-de-trucos-big-o/**
- Big O Cheat Sheet: https://www.bigocheatsheet.com/

Anexos

- Repositorio github: https://github.com/mikesanmartin/UTN-TUPaD-P1-TPI
- Link video youtube: https://youtu.be/eOLtXKCpKnA Asi quedo el video