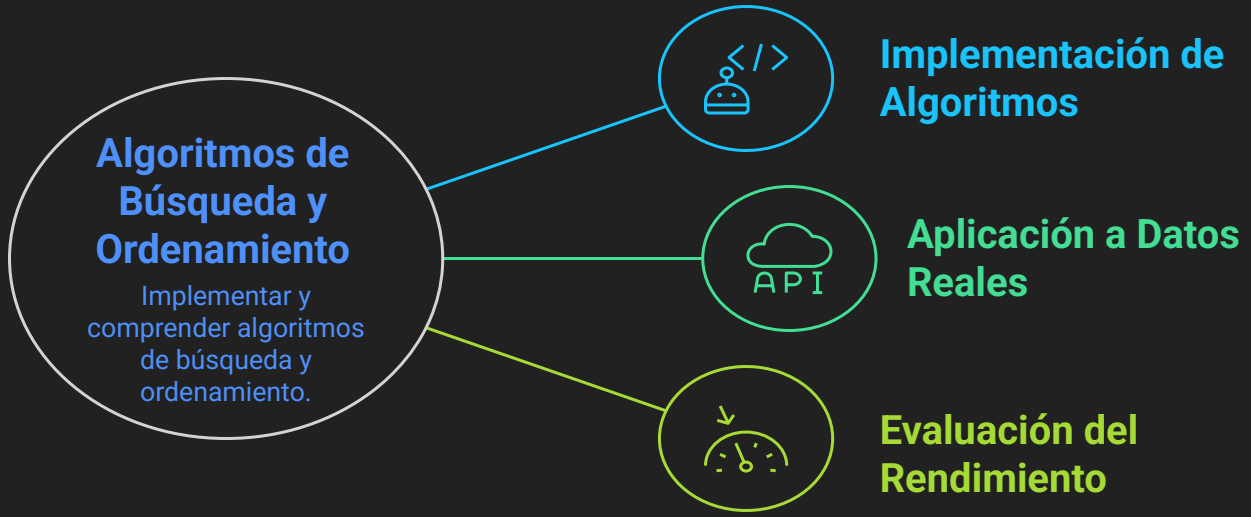
Universidad Tecnológica Nacional

Trabajo Práctico Integrador – Búsqueda y Ordenamiento:

“Análisis y aplicación de búsqueda y ordenamiento en Python sobre datos de la PokéAPI”

Arturo Kaadú
Belén Yarde Buller

Explorando Algoritmos con Datos de Pokémon



Marco Teórico

Búsqueda Lineal

Un método de búsqueda simple que recorre secuencialmente una lista.

Complejidad temporal:

Peor caso: O(n)

Mejor caso: O(1)

Caso promedio: O(n)

Búsqueda Binaria

Un algoritmo de búsqueda eficiente que requiere una lista ordenada.

Complejidad temporal:

Peor caso: O(log n)

Mejor caso: O(1)

Caso promedio: O(log n)

API y Librerías

Herramientas que facilitan la comunicación y el análisis de datos.

PokèAPI/

```
requests
timeit
pprint
```

Marco Teórico

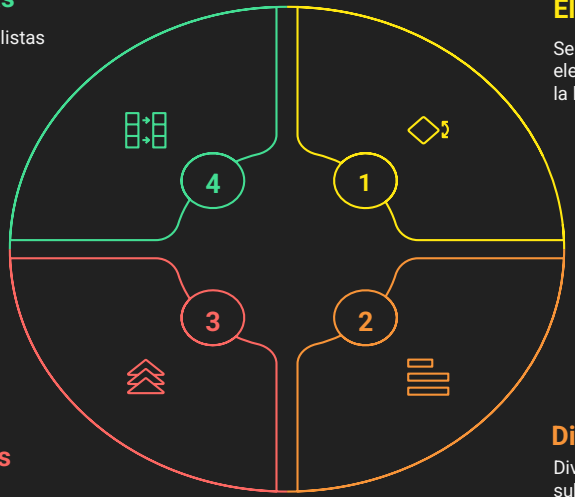
Quick Sort

Combinar Sublistas

Unir las sublistas ordenadas.

Elegir Pivote

Seleccionar un elemento pivote de la lista.



Ordenar Sublistas

Ordenar recursivamente cada sublista.

Dividir Lista

Dividir la lista en sublistas basadas en el pivote.

Bubble Sort

5 Lista Ordenada

Logra una lista completamente ordenada de diccionarios de Pokémon.

4 Reducir Lista

Reduce la porción desordenada de la lista en cada iteración.

3 Burbujear Elemento

Mueve el elemento más grande (o más pequeño) a su posición final.

2 Intercambiar Elementos

Intercambia elementos si no están en el orden deseado.

1 Comparar Pares

Comienza comparando pares adyacentes de elementos en la lista.

Caso Práctico

Ciclo de Gestión de Pokémon

Obtener Pokémon

Recuperar datos de Pokémon de PokéAPI

Buscar Pokémon

Encontrar Pokémon por nombre o altura

Ordenar Pokémon

Organizar Pokémon por nombre o altura

Medir Rendimiento

Evaluar la eficiencia del algoritmo

Metodología

Fase de Investigación

Investigación de algoritmos y recursos

Desarrollo Inicial

Creación de código base

Desarrollo Paralelo

Desarrollo simultáneo en ramas

Pruebas de Rendimiento

Evaluaciones de rendimiento de los algoritmos

Selección de Herramientas

Selección de herramientas de desarrollo

División del Trabajo

División del trabajo en ramas

Integración de Código

Integración de ramas en la principal

Pruebas:
Algoritmos de búsqueda por nombre

```
--- Ejecutando Búsqueda Lineal por Nombre: ---

'arbok' encontrado: {'name': 'arbok', 'height': 35}
Tiempo de ejecución para linear_search por nombre: 0.0006060840096324682 segundos

--- Ejecutando Búsqueda Binaria por Nombre: ---

'arbok' encontrado por Nombre: {'name': 'arbok', 'height': 35}
Tiempo de ejecución para binary_search: 0.00030899999546818435 segundos
```

```
--- Ejecutando Búsqueda Lineal por Nombre: ---

'dodrio' encontrado: {'name': 'dodrio', 'height': 18}
Tiempo de ejecución para linear_search por nombre: 0.003217374993255362 segundos

--- Ejecutando Búsqueda Binaria por Nombre: ---

'dodrio' encontrado por Nombre: {'name': 'dodrio', 'height': 18}
Tiempo de ejecución para binary_search: 0.0007408750243484974 segundos


--- Fin de la ejecución de la opción 1 ---
```

```
--- Ejecutando Búsqueda Lineal por Nombre: ---

'charmander' encontrado: {'name': 'charmander', 'height': 6}
Tiempo de ejecución para linear_search por nombre: 0.0005455419886857271 segundos

--- Ejecutando Búsqueda Binaria por Nombre: ---

'charmander' encontrado por Nombre: {'name': 'charmander', 'height': 6}
Tiempo de ejecución para binary_search: 0.0013169999874662608 segundos
```

Búsquedas por nombre

Elemento buscado	Total de elementos	Búsqueda Lineal	Búsqueda Binaria	Observaciones
"arbok"	30	0.000606s	0.000308s	La búsqueda binaria fue dos veces más rápida que la lineal.
				"arbok" representa un escenario cercano al "peor caso" para la lineal.
"dodrio"	100	0.003217s	0.000740s	La ventaja de la búsqueda binaria se acentúa con el incremento del tamaño de la lista.
"charmander"	100	0.000545s	0.001316s	La búsqueda lineal fue más rápida en su "mejor caso" (elemento cercano al inicio).

Pruebas:
Algoritmos de búsqueda por altura

```
Ingrese la operacion que desea realizar:
0. Salir
1. Busqueda por nombre
2. Busqueda por altura
3. Ordenamiento por altura
4. Ordenamiento por nombre: 2
Por favor, ingrese la altura que desea buscar (en decímetros): 10


--- Ejecutando Búsqueda Lineal por Altura: ---

Pokémon con altura '10.0' encontrado: {'name': 'ivysaur', 'height': 10}
Tiempo de ejecución para linear_search por altura: 0.0001435000158380717 segundos

--- Ejecutando Búsqueda Binaria por Altura: ---

'10' encontrado por Altura: {'name': 'beedrill', 'height': 10}
Tiempo de ejecución para binary_search: 0.0002930420159827918 segundos
```

- Tamaño de lista: 30 elementos
- Valor buscado: 10 decímetros
- Búsqueda lineal levemente más rápida
- Beneficio de escenario de mejor caso posible:

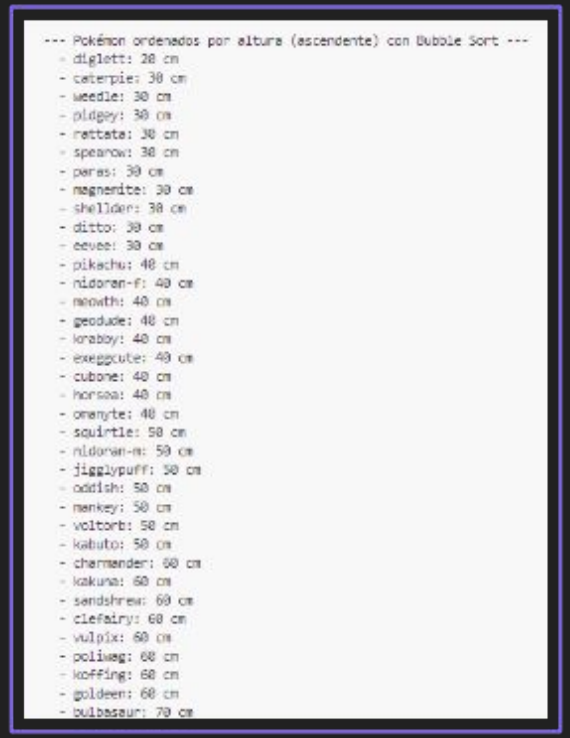
```
➜  tp-integrador-programacion git:(rama-bu
Obteniendo pokemons...
Listado de pokemons obtenido exitosamente
[{'height': 7, 'name': 'bulbasaur'},
 {'height': 10, 'name': 'ivysaur'},
 {'height': 20, 'name': 'venusaur'},
 {'height': 6, 'name': 'charmander'},
 {'height': 11, 'name': 'charmeleon'},
 {'height': 17, 'name': 'charizard'},
 {'height': 5, 'name': 'squirtle'},
```

```
Ingrese la operacion que desea realizar:
0. Salir
1. Busqueda por nombre
2. Busqueda por altura
3. Ordenamiento por altura
4. Ordenamiento por nombre: 2
Por favor, ingrese la altura que desea buscar (en decímetros): 4


--- Ejecutando Búsqueda Lineal por Altura: ---

Pokémon con altura '4.0' encontrado: {'name': 'pikachu', 'height': 4}
Tiempo de ejecución para linear_search por altura: 0.0006747920124325901 segundos

--- Ejecutando Búsqueda Binaria por Altura: ---

'4' encontrado por Altura: {'name': 'nidoran-f', 'height': 4}
Tiempo de ejecución para binary_search: 0.0003102080080158952 segundos
```

- Tamaño de lista: 30 elementos
- Valor buscado: 4 decímetros
- Búsqueda binaria visiblemente más rápida
- Búsqueda lineal afectada por escenario cercano al peor caso

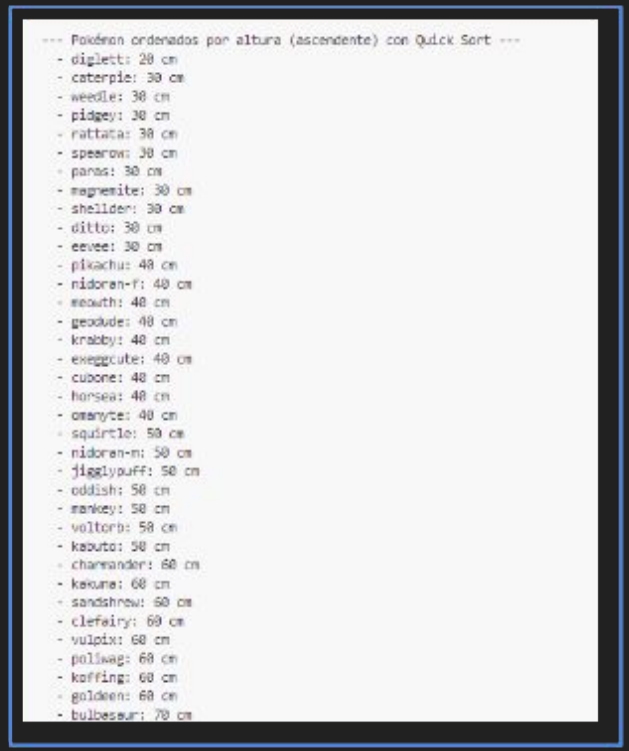
```
 {'height': 35, 'name': 'arbok'},
 {'height': 4, 'name': 'pikachu'},
 {'height': 8, 'name': 'raichu'},
 {'height': 6, 'name': 'sandshrew'},
 {'height': 10, 'name': 'sandslash'},
 {'height': 4, 'name': 'nidoran-f'},
 {'height': 8, 'name': 'nidorina'}]
Ingrese la operacion que desea realizar:
```

Resultados obtenidos. Ordenamiento por altura 150 Pokemon



```
--- Pokémon ordenados por altura (ascendente) con Bubble Sort ---
- diglett: 20 cm
- caterpie: 30 cm
- weedle: 30 cm
- pidgey: 30 cm
- rattata: 30 cm
- spearow: 30 cm
- paras: 30 cm
- magnemite: 30 cm
- shellder: 30 cm
- ditto: 30 cm
- eevee: 30 cm
- pikachu: 40 cm
- nidoran-f: 40 cm
- meowth: 40 cm
- geodude: 40 cm
- krabby: 40 cm
- exeggcute: 40 cm
- cubone: 40 cm
- horsea: 40 cm
- omanyte: 40 cm
- squirtle: 50 cm
- nidoran-m: 50 cm
- jigglypuff: 50 cm
- oddish: 50 cm
- mankey: 50 cm
- voltorb: 50 cm
- kabuto: 50 cm
- charmander: 60 cm
- kakuna: 60 cm
- sandshrew: 60 cm
- clefairy: 60 cm
- vulpix: 60 cm
- poliwag: 60 cm
- koffing: 60 cm
- goldeen: 60 cm
- bulbasaur: 70 cm
```

Tiempo de ejecución para bubble_sort: 0.0016162999672815204 segundos



```
--- Pokémon ordenados por altura (ascendente) con Quick Sort ---
- diglett: 20 cm
- caterpie: 30 cm
- weedle: 30 cm
- pidgey: 30 cm
- rattata: 30 cm
- spearow: 30 cm
- paras: 30 cm
- magnemite: 30 cm
- shellder: 30 cm
- ditto: 30 cm
- eevee: 30 cm
- pikachu: 40 cm
- nidoran-f: 40 cm
- meowth: 40 cm
- geodude: 40 cm
- krabby: 40 cm
- exeggcute: 40 cm
- cubone: 40 cm
- horsea: 40 cm
- omanyte: 40 cm
- squirtle: 50 cm
- nidoran-m: 50 cm
- jigglypuff: 50 cm
- oddish: 50 cm
- mankey: 50 cm
- voltorb: 50 cm
- kabuto: 50 cm
- charmander: 60 cm
- kakuna: 60 cm
- sandshrew: 60 cm
- clefairy: 60 cm
- vulpix: 60 cm
- poliwag: 60 cm
- koffing: 60 cm
- goldeen: 60 cm
- bulbasaur: 70 cm
```

Tiempo de ejecución para quick_sort: 0.0001122000394389033 segundos

Comparación de Algoritmos de Ordenamiento por Tiempo de Ejecución

	Quick Sort	Bubble Sort
Tiempo de Ejecución	0.00011 segundos	0.00161 segundos
Velocidad Relativa	14x más rápido	14x más lento
Eficiencia	Más eficiente	Menos eficiente

Resultados obtenidos. Ordenamiento por altura 30 Pokemon

```
  - squirtle: 50 cm
  - charmander: 60 cm
  - kakuna: 60 cm
  - sandshrew: 60 cm
  - bulbasaur: 70 cm
  - metapod: 70 cm
  - raticate: 70 cm
  - raichu: 80 cm
  - nidorina: 80 cm
  - ivysaur: 100 cm
  - wartortle: 100 cm
  - beedrill: 100 cm
  - sandslash: 100 cm
  - charmeleon: 110 cm
  - butterfree: 110 cm
  - pidgeotto: 110 cm
  - fearow: 120 cm
  - pidgeot: 150 cm
  - blastoise: 160 cm
  - charizard: 170 cm
  - venusaur: 200 cm
  - ekans: 200 cm
  - arbok: 350 cm
Tiempo de ejecución para quick_sort: 4.8200017772614956e-05 segundos
```

```
  - squirtle: 50 cm
  - charmander: 60 cm
  - kakuna: 60 cm
  - sandshrew: 60 cm
  - bulbasaur: 70 cm
  - metapod: 70 cm
  - raticate: 70 cm
  - raichu: 80 cm
  - nidorina: 80 cm
  - ivysaur: 100 cm
  - wartortle: 100 cm
  - beedrill: 100 cm
  - sandslash: 100 cm
  - charmeleon: 110 cm
  - butterfree: 110 cm
  - pidgeotto: 110 cm
  - fearow: 120 cm
  - pidgeot: 150 cm
  - blastoise: 160 cm
  - charizard: 170 cm
  - venusaur: 200 cm
  - ekans: 200 cm
  - arbok: 350 cm
Tiempo de ejecución para bubble_sort: 8.420000085607171e-05 segundos
```

Bubble Sort vs Quick Sort

	 Bubble Sort	 Quick Sort
Tiempo de ejecución	0.0000842 segundos	0.0000482 segundos
Velocidad relativa	1x	1.75x
Eficiencia	Menos eficiente	Más eficiente

Resultados obtenidos. Ordenamiento por nombre: 150 Pokemon

- sandshrew: 60 cm
- sandslash: 100 cm
- scyther: 150 cm
- seadra: 120 cm
- seaking: 130 cm
- seel: 110 cm
- shellder: 30 cm
- slowbro: 160 cm
- slowpoke: 120 cm
- snorlax: 210 cm
- spearow: 30 cm
- squirtle: 50 cm
- starmie: 110 cm
- staryu: 80 cm
- tangela: 100 cm
- tauros: 140 cm
- tentacool: 90 cm
- tentacruel: 160 cm
- vaporeon: 100 cm
- venomoth: 150 cm
- venonat: 100 cm
- venusaur: 200 cm
- victreebel: 170 cm
- vileplume: 120 cm
- voltorb: 50 cm
- vulpix: 60 cm
- wartortle: 100 cm
- weedle: 30 cm
- weepinbell: 100 cm
- weezing: 120 cm
- wigglytuff: 100 cm
- zapdos: 160 cm
- zubat: 80 cm

Tiempo de ejecución para quick_sort: 0.00020819995552301407 segundos

- seadra: 120 cm
- seaking: 130 cm
- seel: 110 cm
- shellder: 30 cm
- slowbro: 160 cm
- slowpoke: 120 cm
- snorlax: 210 cm
- spearow: 30 cm
- squirtle: 50 cm
- starmie: 110 cm
- staryu: 80 cm
- tangela: 100 cm
- tauros: 140 cm
- tentacool: 90 cm
- tentacruel: 160 cm
- vaporeon: 100 cm
- venomoth: 150 cm
- venonat: 100 cm
- venusaur: 200 cm
- victreebel: 170 cm
- vileplume: 120 cm
- voltorb: 50 cm
- vulpix: 60 cm
- wartortle: 100 cm
- weedle: 30 cm
- weepinbell: 100 cm
- weezing: 120 cm
- wigglytuff: 100 cm
- zapdos: 160 cm
- zubat: 80 cm

Tiempo de ejecución para bubble_sort: 0.0025790000217966735 segundos

Comparación de Rendimiento: Quick Sort vs. Bubble Sort

	Quick Sort	Bubble Sort
Velocidad	0.00020 segundos	0.00257 segundos
Rendimiento Relativo	12.3x más rápido	12.3x más lento

Tiempos de ejecución de algoritmos

Conclusiones

Búsqueda

La búsqueda binaria es más eficiente para listas grandes. La búsqueda lineal es más rápida si

Ordenamiento

Quick Sort es consistentemente más rápido que Bubble Sort.

Aprendizaje General

Este proyecto nos ayudó a entender la complejidad de los algoritmos.

Mejoras Futuras

Mejorar los tiempos de carga utilizando llamadas a API asíncronas.

Utilidad

Este conocimiento es vital para elegir el algoritmo correcto.

¡Gracias por su atención!