

Trabajo Integrador – Programación I

Algoritmos de Búsqueda y Ordenamiento en Python

Alumno:

Ignacio Robles – ignaciomrobles2@gmail.com

Materia:

Programación I

Profesores:

Oscar Londero

Cinthia Rigoni

Fecha de entrega:

09/06/2025

Índice

1. Introducción
 2. Marco Teórico
 3. Caso Práctico
 4. Metodología Utilizada
 5. Resultados Obtenidos
 6. Conclusiones
 7. Bibliografía
 8. Anexos
-

1. Introducción

En el presente trabajo se aborda el tema de **algoritmos de búsqueda y ordenamiento en Python**, seleccionando dos de los más representativos y accesibles para la comprensión inicial de estructuras algorítmicas: **Insertion Sort** (ordenamiento por inserción) y **Búsqueda Lineal**.

Se eligió este tema debido a que estos algoritmos son herramientas básicas en programación y constituyen la base para resolver problemas que requieren organización o localización de datos. Son además esenciales para desarrollar lógica algorítmica y estructuración de código.

El objetivo principal es demostrar el funcionamiento de estos algoritmos mediante una **aplicación práctica sencilla**, mostrando de forma clara sus ventajas y limitaciones, y consolidar los conocimientos teóricos con un proyecto concreto.

2. Marco Teórico

Un algoritmo es un conjunto de instrucciones o pasos definidos que permiten resolver un problema o realizar una tarea. Los algoritmos de búsqueda y ordenamiento son fundamentales en la programación y permiten organizar o localizar datos en estructuras como listas o arrays.

El algoritmo de ordenamiento utilizado es el Insertion Sort, que funciona insertando elementos en una posición correcta dentro de una lista parcialmente ordenada. Es eficiente para listas pequeñas o cuando los datos ya están parcialmente ordenados.

Por otro lado, el algoritmo de búsqueda lineal permite encontrar un elemento recorriendo secuencialmente cada posición de la lista, comparando uno por uno hasta hallar coincidencia o terminar el recorrido.

Ambos algoritmos permiten consolidar conceptos de estructuras secuenciales, condicionales y ciclos en Python.

3. Caso Práctico

Descripción del problema

Se plantea el desarrollo de un programa que gestiona una lista de canciones de un álbum musical. Cada canción cuenta con un número de pista, nombre y duración.

El programa debe cumplir con las siguientes funcionalidades:

- Ordenar las canciones **por número de pista** o **por duración** utilizando **Insertion Sort**.
- Buscar canciones por **número de pista** utilizando **búsqueda lineal**.
- Mostrar los resultados al usuario de manera clara en consola.

Código fuente comentado

Programa: Gestión de canciones del álbum "Addison" - Addison Rae

Tema: Algoritmos de Ordenamiento (Insertion Sort) y Búsqueda (Lineal)

Autor: Ignacio Robles

Fecha: 09/06/2025

Lista de canciones del álbum "Addison" con el formato:

(número de track, nombre de la canción, duración en minutos)

canciones = [

(1, "New York", 2.32),

(2, "Diet Pepsi", 2.49),

(3, "Money Is Everything", 2.02),

(4, "Aquamarine", 2.42),

(5, "Lost & Found", 0.48),

(6, "High Fashion", 3.18),

(7, "Summer Forever", 3.47),
(8, "In the Rain", 3.33),
(9, "Fame Is a Gun", 3.03),
(10, "Times Like These", 3.52),
(11, "Life's No Fun Through Clear Waters", 0.57),
(12, "Headphones On", 4.00),
]

```
def insertion_sort(lista, key=lambda x: x):
```

```
    """
```

Algoritmo de ordenamiento por inserción (Insertion Sort).

Parámetros:

- lista: lista de elementos a ordenar.
- key: función opcional que define el criterio de ordenamiento.

Retorna:

- Una nueva lista ordenada según el criterio definido.

```
    """
```

```
    for i in range(1, len(lista)):
```

```
        actual = lista[i]
```

```
        j = i - 1
```

```
        # Desplaza elementos mayores hacia adelante
```

```
        while j >= 0 and key(lista[j]) > key(actual):
```

```
        lista[j + 1] = lista[j]

        j -= 1

        lista[j + 1] = actual

    return lista
```

```
def busqueda_lineal(lista, numero_track):
```

```
    """
```

Algoritmo de búsqueda lineal.

Busca una canción por su número de track en la lista.

Parámetros:

- lista: lista de canciones.
- numero_track: número del track que se desea buscar.

Retorna:

- Una tupla con la información de la canción si se encuentra, o None si no se encuentra.

```
    """
```

```
    for cancion in lista:
```

```
        if cancion[0] == numero_track:
```

```
            return cancion
```

```
    return None
```

Programa principal

Ordenamiento de las canciones por número de track

```
canciones_ordenadas = insertion_sort(canciones.copy(), key=lambda x:  
x[0])
```

```
print("Canciones ordenadas por número de track:")
```

```
for num, nombre, duracion in canciones_ordenadas:
```

```
    print(f"{num}. {nombre} - {duracion} min")
```

Ordenamiento de las canciones por duración

```
print("\nCanciones ordenadas por duración:")
```

```
canciones_por_duracion = insertion_sort(canciones.copy(), key=lambda  
x: x[2])
```

```
for num, nombre, duracion in canciones_por_duracion:
```

```
    print(f"{num}. {nombre} - {duracion} min")
```

Bucle interactivo para buscar canciones por número de track

```
print("\nBuscar canciones por número de track (escriba '0' para salir:)")
```

```
while True:
```

```
    try:
```

```
        numero = int(input("Ingrese el número de track: "))
```

```
        if numero == 0:
```

```
            print("Finalizando búsqueda.")
```

```
            break
```

```
        resultado = busqueda_lineal(canciones, numero)
```

```
        if resultado:
```

```
print(  
    f"El track número {numero} es '{resultado[1]}' con una duración  
de {resultado[2]} minutos."  
  
    else:  
  
        print(f"No se encontró ningún track con el número {numero}.")  
  
except ValueError:  
  
    print("Por favor, ingrese un número válido.")
```

Decisiones de diseño

Se eligió el algoritmo **Insertion Sort** por su simplicidad y porque es adecuado para listas pequeñas como la del ejemplo. Para búsquedas se implementó la **búsqueda lineal**, debido a que es eficiente en listas pequeñas y permite mostrar claramente el recorrido secuencial.

Validación del funcionamiento

Se realizaron múltiples pruebas ingresando diferentes números de pista y visualizando el resultado correcto de la búsqueda y del ordenamiento solicitado.

4. Metodología Utilizada

1. **Investigación previa:** Consultas a fuentes oficiales de Python, YouTube y bibliografía especializada sobre algoritmos.
2. **Diseño del problema:** Definición de estructura de datos adecuada (listas de tuplas).
3. **Implementación del código:** Programado en **Python 3.12** utilizando **Visual Studio Code** como entorno de desarrollo.
4. **Pruebas:** Se realizaron varias pruebas manuales en consola.

5. **Control de versiones:** Se utilizó **Git** y **GitHub** para respaldar el código (enlace en anexos).
-

5. Resultados Obtenidos

- **El ordenamiento por número de pista y por duración funcionó correctamente** utilizando **Insertion Sort**.
 - La búsqueda por número de pista devolvió resultados correctos en todos los casos probados.
 - **Dificultades:** Inicialmente, se intentó utilizar la función `sorted()` de Python, ya que en YouTube se decía que era muy eficiente pero se descartó para implementar el algoritmo manualmente, respetando el objetivo del trabajo.
 - **Repositorio:** <https://github.com/roblesignacio/TP-integrador-programacion-1.git>
-

6. Conclusiones

El proyecto permitió comprobar que los algoritmos **Insertion Sort** y **Búsqueda Lineal** son herramientas eficaces para resolver problemas cotidianos en programación relacionados con listas pequeñas.

Se afianzaron los conceptos de algoritmos, estructuras de datos y lógica secuencial, además del uso del lenguaje Python.

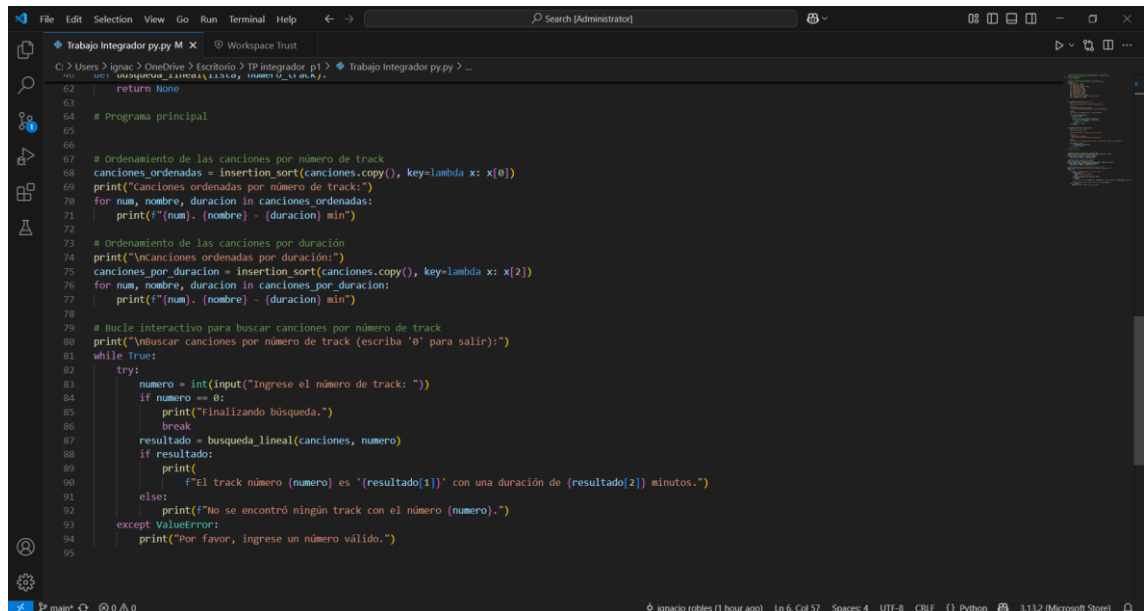
Como posibles mejoras, podría implementarse búsqueda binaria (con listas ordenadas) o algoritmos de ordenamiento más eficientes como **Quicksort** para listas grandes.

7. Bibliografía

- Ordenamiento por inserción de Chio Code
<https://youtu.be/6GU6AGEWYJY?si=Wh7TX1boVsiCwviw>.

- Programiz. (s.f.). Python Insertion Sort.
<https://www.programiz.com/python-programming/examples/insertion-sort>
- Tutorialspoint. (s.f.). Python Linear Search.
<https://www.tutorialspoint.com/python-program-to-perform-linear-search>

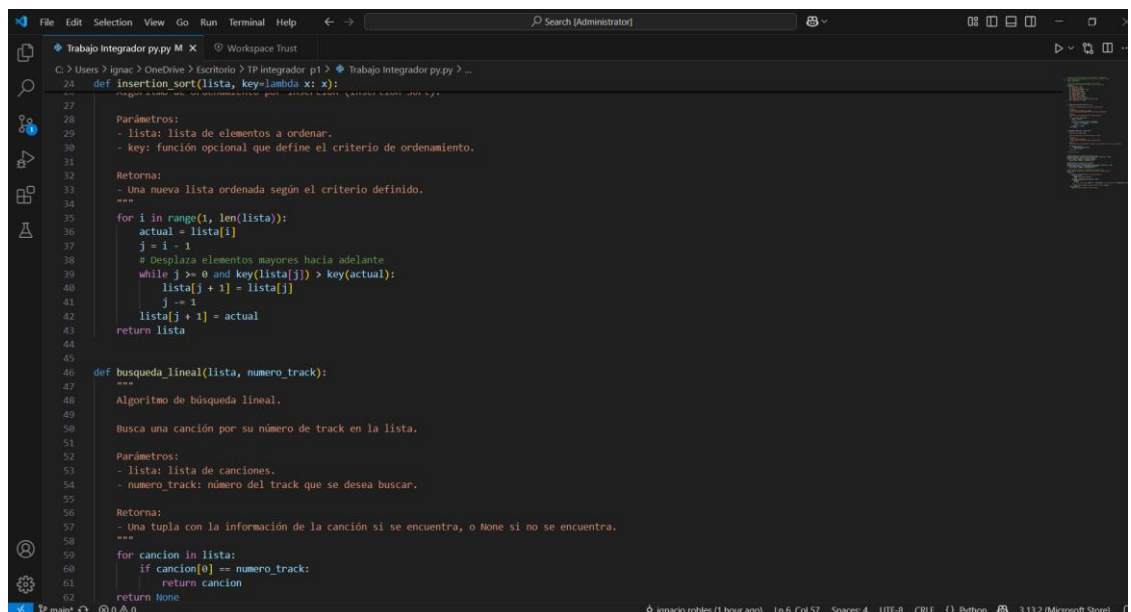
8. Anexos



```

File Edit Selection View Go Run Terminal Help
C:\Users\ignac> OneDrive> Escritorio> TP integrador p1> Trabajo Integrador.py
60 def busqueda_lineal(canciones, numero):
61     return None
62
63
64 # Programa principal
65
66
67 # Ordenamiento de las canciones por número de track
68 cancion_ordenadas = insertion_sort(canciones.copy(), key=lambda x: x[0])
69 print("Canciones ordenadas por número de track:")
70 for num, nombre, duracion in cancion_ordenadas:
71     print(f"{num}. {nombre} - {duracion} min")
72
73 # Ordenamiento de las canciones por duración
74 print("\nCanciones ordenadas por duración:")
75 cancion_por_duracion = insertion_sort(canciones.copy(), key=lambda x: x[2])
76 for num, nombre, duracion in cancion_por_duracion:
77     print(f"{num}. {nombre} - {duracion} min")
78
79 # Bucle interactivo para buscar canciones por número de track
80 print("\nBuscar canciones por número de track (escriba '0' para salir):")
81 while True:
82     try:
83         numero = int(input("Ingrese el número de track: "))
84         if numero == 0:
85             print("Finalizando búsqueda.")
86             break
87         resultado = busqueda_lineal(canciones, numero)
88         if resultado:
89             print(
90                 f"El track número {numero} es '{resultado[1]}' con una duración de {resultado[2]} minutos."
91             )
92         else:
93             print(f"No se encontró ningún track con el número {numero}.")
94     except ValueError:
95         print("Por favor, ingrese un número válido.")

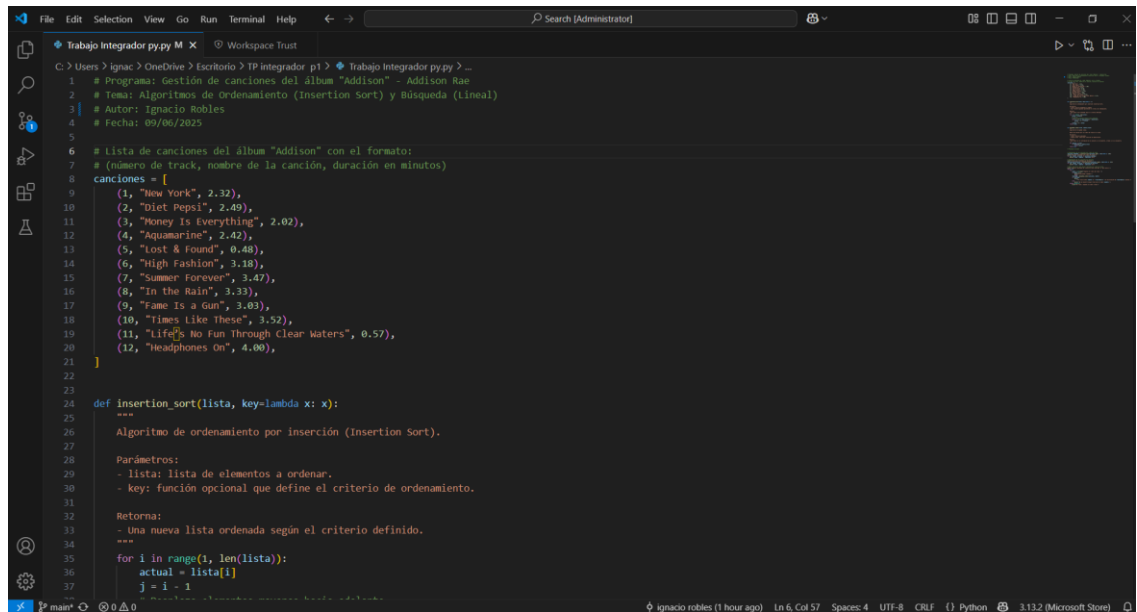
```



```

File Edit Selection View Go Run Terminal Help
C:\Users\ignac> OneDrive> Escritorio> TP integrador p1> Trabajo Integrador.py
24 def insertion_sort(lista, key=lambda x: x):
25     """
26     Algoritmo de ordenamiento por inserción.
27     """
28     Parámetros:
29     - lista: lista de elementos a ordenar.
30     - key: función opcional que define el criterio de ordenamiento.
31
32     Retorna:
33     - Una nueva lista ordenada según el criterio definido.
34     """
35     for i in range(1, len(lista)):
36         actual = lista[i]
37         j = i - 1
38         # Desplaza elementos mayores hacia adelante
39         while j >= 0 and key(lista[j]) > key(actual):
40             lista[j + 1] = lista[j]
41             j -= 1
42         lista[j + 1] = actual
43     return lista
44
45
46 def busqueda_lineal(lista, numero_track):
47     """
48     Algoritmo de búsqueda lineal.
49
50     Busca una canción por su número de track en la lista.
51
52     Parámetros:
53     - lista: lista de canciones.
54     - numero_track: número del track que se desea buscar.
55
56     Retorna:
57     - Una tupla con la información de la canción si se encuentra, o None si no se encuentra.
58     """
59     for cancion in lista:
60         if cancion[0] == numero_track:
61             return cancion
62     return None

```



```
1 # Programa: Gestión de canciones del álbum "Addison" - Addison Rae
2 # Tema: Algoritmos de Ordenamiento (Insertion Sort) y Búsqueda (Lineal)
3 # Autor: Ignacio Robles
4 # Fecha: 09/06/2025
5
6 # Lista de canciones del álbum "Addison" con el formato:
7 # (número de track, nombre de la canción, duración en minutos)
8 canciones = [
9     (1, "New York", 2.32),
10    (2, "Diet Pepsi", 2.49),
11    (3, "Money Is Everything", 2.02),
12    (4, "Aquamarine", 2.42),
13    (5, "Lost & Found", 0.48),
14    (6, "High Fashion", 3.18),
15    (7, "Summer Forever", 3.47),
16    (8, "In the Rain", 3.33),
17    (9, "Fame Is a Gun", 3.03),
18    (10, "Times Like These", 3.52),
19    (11, "Life's No Run Through Clear Waters", 0.57),
20    (12, "Headphones On", 4.80),
21 ]
22
23
24 def insertion_sort(lista, key=lambda x: x):
25     """
26     Algoritmo de ordenamiento por inserción (Insertion Sort).
27
28     Parámetros:
29     - lista: lista de elementos a ordenar.
30     - key: función opcional que define el criterio de ordenamiento.
31
32     Retorna:
33     - Una nueva lista ordenada según el criterio definido.
34     """
35     for i in range(1, len(lista)):
36         actual = lista[i]
37         j = i - 1
38         while j >= 0 and lista[j] > actual:
39             lista[j+1] = lista[j]
40             j -= 1
41         lista[j+1] = actual
```

- Video explicativo: https://drive.google.com/file/d/1WTCAfSS4-R3dqUwLD5hH114j_mxkStrW/view?usp=drive_link
- <https://github.com/roblesignacio/TP-integrador-programacion-1.git>