

# 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 finito y ordenado de pasos que resuelven un problema o realizan una tarea específica. En programación, los algoritmos permiten procesar datos, tomar decisiones, repetir acciones y obtener resultados en función de condiciones dadas. Dentro de este amplio campo, los algoritmos de **búsqueda y ordenamiento** son fundamentales para la eficiencia de cualquier programa.

### Algoritmos de Ordenamiento

Ordenar una lista significa reorganizar sus elementos según un criterio (por ejemplo, de menor a mayor). Existen muchos algoritmos de ordenamiento, pero en este trabajo se utiliza **Insertion Sort (ordenamiento por inserción)**. Este método recorre la lista elemento por elemento y lo va insertando en la posición correcta dentro de una sublista ya ordenada.

Insertion Sort tiene una **complejidad temporal de  $O(n^2)$**  en el peor caso, pero puede ser muy eficiente para listas pequeñas o ya parcialmente ordenadas.

Además, es estable (no cambia el orden de elementos iguales) y fácil de implementar, lo que lo hace ideal para fines educativos.

## Algoritmos de Búsqueda

Buscar consiste en localizar un elemento dentro de una colección. La **búsqueda lineal** (o secuencial) es la más simple: recorre la lista desde el inicio y compara cada elemento con el buscado hasta encontrarlo o llegar al final.

Aunque no es la más eficiente (su complejidad es también  $O(n)$ ), tiene la ventaja de que funciona en listas **no ordenadas**, a diferencia de otros algoritmos como la búsqueda binaria, que requieren orden previo.

## Relación con Python y la Programación

Implementar estos algoritmos en Python permite poner en práctica conceptos clave de la programación como:

- **Estructuras de datos:** listas y tuplas.
- **Control de flujo:** estructuras condicionales y bucles.
- **Funciones:** encapsulamiento y reutilización de lógica.
- **Parámetros y funciones lambda:** para personalizar el criterio de ordenamiento.
- **Manejo de errores con try/except:** en el caso de entrada de datos inválida.

Estas estructuras son pilares de cualquier lenguaje de programación moderno y fundamentales para resolver problemas reales.

---

## 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.

---

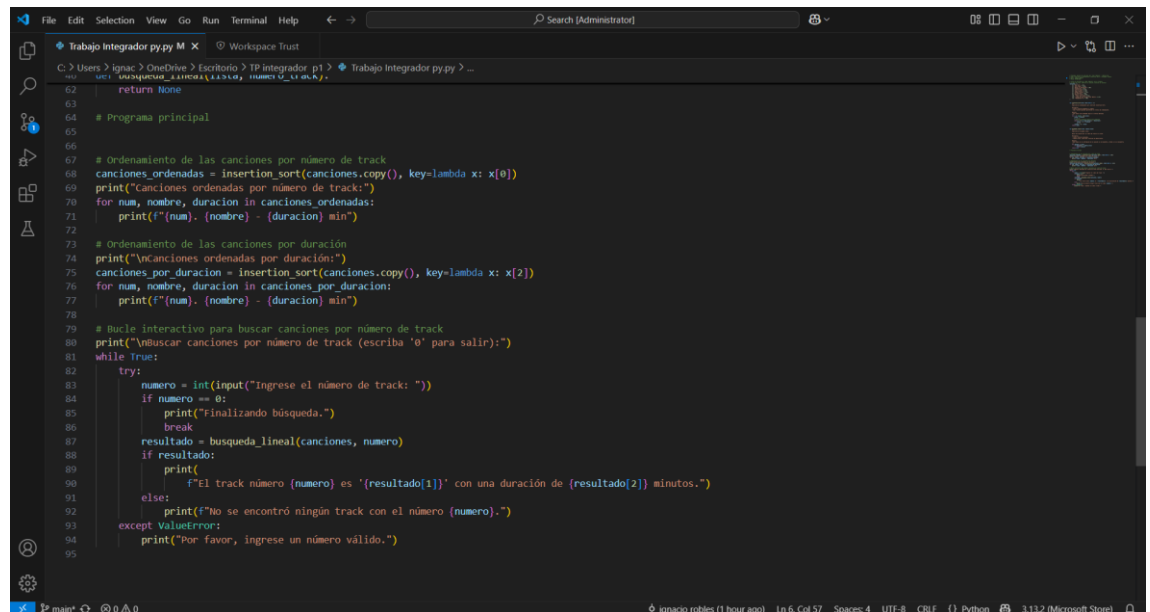
## 7. Bibliografía

- 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>

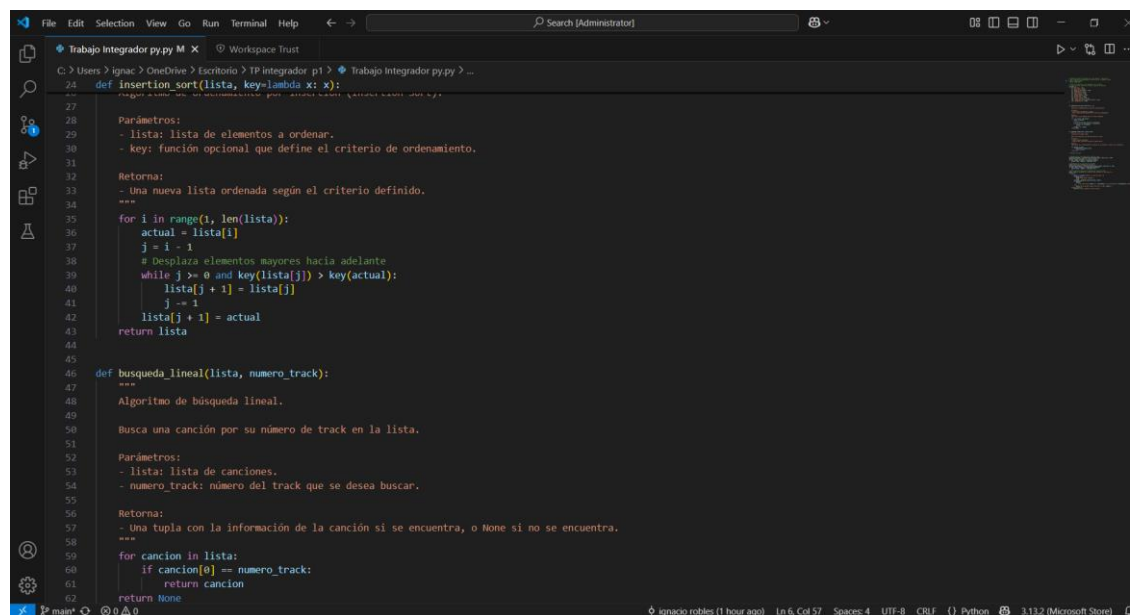
## 8. Anexos



```

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

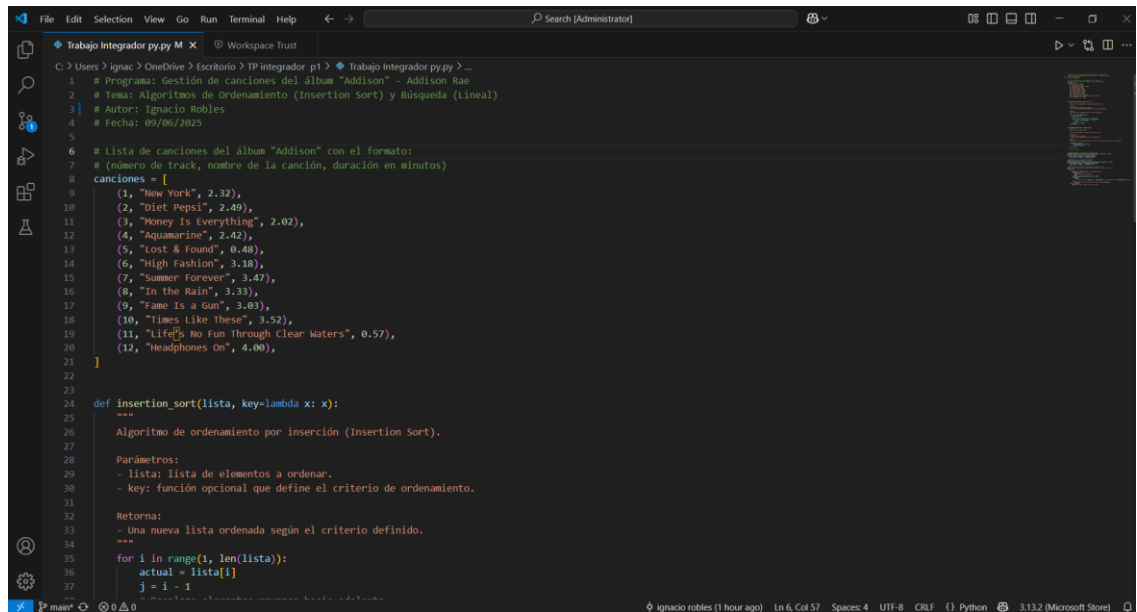
```



```

File Edit Selection View Go Run Terminal Help
Trabajo Integrador py.py M X Workspace Trust
C:\Users\ignac> ignac> OneDrive> Escritorio> TP Integrador p1> Trabajo Integrador py.py> ...
24 def insertion_sort(lista, key=lambda x: x):
25     """
26     Ordena una lista de elementos según el criterio definido.
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", 4.00),
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://drive.google.com/file/d/1WTCAfSS4-R3dqUwLD5hH114j_mxkStrW/view?usp=drive_link)
- <https://github.com/roblesignacio/TP-integrador-programacion-1.git>