

Trabajo Práctico Integrador:

Algoritmos de Búsqueda y Ordenamiento

Alumnos:

Córdoba Natalia- nataliacorrdoba@gmail.com

Duarte Federico- fededuartefilo1@gmail.com

Materia:

Programación I

Profesor:

Bruselario, Sebastián

Comisión:

7

Fecha de Entrega:

9 de junio de 2025

Índice

1. Introducción	3
2. Marco teórico	4
3. Caso práctico	7
4. Metodología utilizada	12
5. Resultados obtenidos	13
6. Conclusiones	14
7. Bibliografía	15
8. Anexos	16

1. Introducción

Este trabajo analiza el uso de algoritmos de búsqueda y de ordenamiento. Este tema fue seleccionado por su gran importancia en la programación, debido a que se encuentran presentes en gran variedad de tareas como el manejo de bases de datos, optimización de procesos, inteligencia artificial, entre otras. De este modo su uso y conocimiento es fundamental para cualquier programador .

Estos algoritmos son herramientas muy importantes para el desarrollo de software, ya que permiten a los programadores obtener respuestas más eficientes, escalables y precisas a la hora de gestionar información. La utilización de ambos tipos de algoritmos contribuye a que los desarrolladores generen programas mucho más organizados, rápidos y confiables. De modo tal que es fundamental el conocimiento de estos algoritmos a la hora de programar dado que cuando una base de datos aumenta su información las exigencias de los componentes también lo hacen, tal que para mantener la estabilidad del programa es necesario que se tengan en cuenta los mejores métodos para su funcionamiento.

Mediante la realización de este trabajo, se espera alcanzar una serie de objetivos. En primer lugar, lograr el aprendizaje teórico sobre los algoritmos de búsqueda y de ordenamiento. En segundo lugar, usar estos conceptos en un proyecto práctico real combinándolos con los otros conocimientos en programación que se adquirieron durante la cursada de esta cátedra. Y en tercer lugar, comprender la utilidad que tienen estos algoritmos para un futuro técnico en programación.

2. Marco teórico

A la hora de hacer búsquedas en bases de datos un programador debe contar con el conocimiento de ciertos algoritmos para que esta tarea sea lo más óptima para realizar su función, en este trabajo nos centramos en los siguientes algoritmos de búsqueda:

- *Algoritmo de búsqueda lineal:* Este es un algoritmo de los más clásicos que recorre la lista desde el principio buscando su objetivo de a un espacio hasta encontrarlo o bien, puede suceder que su objetivo esté en el último espacio de la lista y tener que recorrerla toda. Por eso está bien para una lista pequeña o bien a sabiendas de que el valor que se busca está al principio de la misma. Pero por otro lado es bastante endeble a la hora de encontrar un dato en una lista muy larga, dado que la recorre de a un elemento.
- *Algoritmo de búsqueda binaria:* Este algoritmo va dividiendo la lista para recorrerla y buscar el datos de modo tal que la lista se separa, es útil en casos en los cuales las listas son más extensas. La particularidad que tiene este tipo de algoritmo de búsqueda es que funciona solo en listas ordenadas.

La elección de qué algoritmo de búsqueda es más conveniente depende de las características del programa que se esté desarrollando. Pero principalmente esta decisión está relacionada al tamaño de la lista en la cual se deba realizar la búsqueda. Si es una lista pequeña, la búsqueda lineal es una excelente opción, sin embargo, si se está trabajando con una lista muy extensa, el algoritmo más eficiente será la búsqueda binaria.

Además de realizar búsquedas, otro conocimiento importante que debe tener un programador, es saber ordenar en base a ciertos criterios los datos con los que se está trabajando. Para ello, se utilizan algoritmos de ordenamientos que permiten organizar los datos para poder trabajar con ellos de forma más efectiva. Ya que si los datos se encuentran ordenados otras tareas, que se deban ejecutar sobre estos se podrán realizar más rápidamente. Los algoritmos de ordenamientos que se utilizarán para el desarrollo de este trabajo son los siguientes:

- *Ordenamiento por burbuja o Bubble Sort:* Este algoritmo es muy sencillo de comprender pero no resulta eficiente si las listas con las que se trabaja son

muy extensas. La forma en la que funciona es a través de la comparación de cada elemento de la lista con el elemento que le sigue. Si estos elementos no están en el orden correcto se intercambian de lugar. Se continua con este proceso hasta terminar con todos los ítems de la lista.

- *Ordenamiento por selección o Selection Sort:* El método que utiliza este algoritmo para realizar el ordenamiento es buscar el elemento más pequeño de la lista y ubicarlo al inicio, repitiendo hasta llegar al último elemento. Este también es un método sencillo, por lo tanto tampoco es el más conveniente cuando se tienen listas grandes pero es un poco más eficiente que el algoritmo bubble sort.
- *Ordenamiento por inserción o Insertion Sort:* Este algoritmo funciona construyendo la lista ordenada de a un elemento a la vez, insertando cada uno en la posición que le corresponde. De la misma manera que los algoritmos anteriores, este método es eficiente en listas pequeñas pero también en listas que estén en parte ordenadas.
- *Ordenamiento rápido o Quick Sort:* Este método consiste en tomar un elemento para que funcione como un pivote, luego dividir la lista en dos partes para que los elementos menores al pivote queden a la izquierda y los mayores a la derecha. Se continúa aplicando de forma recursiva este proceso en cada una de las partes hasta lograr que la lista quede completamente ordenada. Su funcionamiento es mucho más rápido que Bubble Sort.

Al igual que con los algoritmos de búsqueda, seleccionar cuál es el algoritmo de ordenamiento ideal para utilizar depende de distintos factores que el programador va a tener que analizar, como por ejemplo el tamaño de la lista o el tipo de datos.

Utilizar algoritmos de búsqueda y de ordenamiento, es una gran decisión a la hora de desarrollar software, ya que hay una serie de beneficios que se pueden obtener al implementarlos. En relación a la eficiencia y precisión, permiten mejorar en gran medida el tiempo de ejecución de los programas específicamente cuando la cantidad de datos es muy grande, garantizando que la recuperación se haga de forma correcta. Por otro lado, en cuanto a la organización facilitan el trabajo con los datos ya que ofrecen la posibilidad de trabajar con ellos de forma más ordenada, lo que hace más simple su análisis posterior. Asimismo, en relación a la escalabilidad y

versatilidad, estos algoritmos ofrecen grandes beneficios ya que permiten trabajar con distintas cantidades de datos adaptándose a las necesidades del programa que se esté desarrollando así como ser utilizadas en diferentes contextos.

3. Caso práctico

Con el fin de poder poner en práctica los algoritmos de búsqueda y ordenamiento, se creó un juego interactivo en Python. La idea del programa consistió en probar la ejecución de ciertos algoritmos de búsqueda y ordenamiento de una manera más entretenida que nos permitiera, además utilizar otros conceptos aprendidos a lo largo de la cursada. Se utilizaron en el programa los siguientes algoritmos:

- Algoritmos de búsqueda:
 - *Búsqueda Lineal*

```
# FUNCIONES DE BÚSQUEDA
def busqueda_lineal(lista, objetivo):
    for i in range(len(lista)):
        if lista[i] == objetivo:
            return i
    return -1
```

- *Búsqueda Binaria*

```
# funcion busqueda binaria
def busqueda_binaria(lista_binario, objetivo):
    izquierda, derecha = 0, len(lista_binario) - 1
    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        if lista_binario[medio] == objetivo:
            return medio
        elif lista_binario[medio] < objetivo:
            izquierda = medio + 1
        else:
            derecha = medio - 1
    return -1
```

- Algoritmos de ordenamiento:

- *Bubble Sort*

```
# Funcion Bubble_sort
def bubble_sort(lista):
    for i in range(len(lista)):
        for j in range(len(lista) - 1 - i):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1],
lista[j]
    return lista
```

- *Selection Sort*

```
# Funcion Selection
def selection_sort(lista):
    for i in range(len(lista)):
        minimo = i
        for j in range(i + 1, len(lista)):
            if lista[j] < lista[minimo]:
                minimo = j
        lista[i], lista[minimo] = lista[minimo], lista[i]
    return lista
```

- *Insertion Sort*

```
# funcion insertion_sort
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >= 0 and key < arr[j] :
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key
    return arr
```

- *Quick Sort*

```
# funcion quick_sort
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
```



```

less = [x for x in arr[1:] if x <= pivot]
greater = [x for x in arr[1:] if x > pivot]
return quick_sort(less) + [pivot] +
quick_sort(greater)

```

Como juego se realizó un quiz que contiene seis preguntas de opción múltiple, una para cada algoritmo que mencionamos. Cuando comienza el juego, se le da la bienvenida al usuario y luego se presentan una a una las preguntas. El usuario debe responder qué algoritmo utilizaría para resolver un problema que se le presenta. Si elige correctamente, se muestra por consola un mensaje que dice “Correcto” junto con la resolución del problema llamando a la función que corresponde al algoritmo, además se suma un punto a los aciertos. Si el usuario se equivoca, no se llama a la función del algoritmo, solo se muestra un mensaje que dice “incorrecto” y se pasa a la siguiente pregunta. Finalizados los seis niveles se muestran, también por consola, los resultados de la partida, mencionando la cantidad de aciertos realizados y dando un mensaje que varía de acuerdo a la cantidad de respuestas correctas.

El código se estructuró de la siguiente manera:

- *Función para armar listas:* Esta función está pensada para el ejemplo de búsqueda binaria, de modo que muestre mejor su utilidad para listas grandes.

```

def generar_lista():
    return list(range(1,100,3))
lista_binario=generar_lista()

```

- *Preguntas del juego:* Incluye una lista de diccionarios con las preguntas, cada una compuesta por los elementos nivel, texto, opciones y respuesta correcta (solo se deja un caso de ejemplo).

```

PREGUNTAS = [
    # Nivel 1: Búsqueda Lineal
    {"nivel": 1,
     "texto": "Si Tienes la siguiente lista: [5, 3, 8, 2, 9, 1]. \nDebes buscar el número 8 recorriéndola elemento por elemento. ¿Qué algoritmo utilizarías?",
     "opciones": ["Búsqueda lineal", "Búsqueda binaria"],
     "respuesta_correcta": "1"
    },

```

- *Función hacer pregunta:* Aquí se presenta la que va mostrando una a una las preguntas y que llama a las funciones de los algoritmos si el usuario responde correctamente.

```
def hacer_pregunta(pregunta):
    print(f"\nNivel {pregunta['nivel']}:")
    print(f"{pregunta['texto']}")
    for i, opcion in enumerate(pregunta['opciones'], 1):
        print(f"{i}. {opcion}")

    while True:
        respuesta = input(f"Tu respuesta (número): ")
        if respuesta.isdigit() and 1 <= int(respuesta) <= len(pregunta['opciones']):
            break
        print(f";Opción inválida! Ingresas un número entre 1 y {len(pregunta['opciones'])}")

    if respuesta == pregunta['respuesta_correcta']:
        print("="*80)
        print("¡Correcto!")
        resolver_algoritmo(pregunta['nivel'])
        print("="*80)
        return True
    else:
        print(f"Incorrecto. La respuesta correcta era: {pregunta['opciones'][int(pregunta['respuesta_correcta'])-1]}")

    return False
```

- *Funciones Algoritmos de Búsqueda:* Aquí se presentan las funciones de los dos tipos de algoritmos de búsqueda que utilizamos, lineal y binaria, dichas funciones se presentan con anterioridad .
- *Funciones Algoritmos de Ordenamiento:* Colocamos aquí las funciones para los algoritmos de ordenamiento Bubble Sort, Selection Sort, Insertion Sort y Quick Sort, que se muestran anteriormente .
- *Función resolver algoritmo:* Esta función brinda los parámetros que debemos pasarle a cada una de las funciones planteadas para los algoritmos, Esta función puede verse en la sección de anexo al final del documento.

- *Menú principal del Juego:* Acá se creó la función jugar() que se encarga de dar la bienvenida al usuario, llamar a la función hacer_preguntas(), llevar la cuenta de los aciertos y mostrar los resultados al finalizar la partida.

```
def jugar():
    while True:
        print("="*80)
        print("BIENVENIDO/A AL QUIZZ DE ALGORITMOS")
        print("="*80)
        aciertos = 0
        for pregunta in PREGUNTAS:
            if hacer_pregunta(pregunta):
                aciertos += 1
        print("="*80)
        print(f"Juego terminado. Aciertos: {aciertos}")
        if aciertos == 6:
            print(f";Perfecto! Has respondido correctamente todas las preguntas.")
        elif aciertos >= 4:
            print(f";Muy bien! Has respondido bien más de la mitad de las preguntas.")
        else:
            print(f"Debes practicar un poco más.")

        volver_a_jugar= input("\n¿Quieres jugar de nuevo? (sí/no): ").lower().strip()
        if volver_a_jugar != 'si' and volver_a_jugar != 'sí':
            print(";Gracias por jugar! ;Hasta la próxima!")
            break
```

- *Entrada al juego:* En esta última parte, es donde llamamos a la función jugar() para que se ejecute y comience el juego.

```
if __name__ == "__main__": # Ejecuta el juego solo si el
    archivo se ejecuta directamente
    jugar()
```

4. Metodología utilizada

Para la realización de este trabajo se realizaron una serie de pasos:

- Investigación sobre el tema elegido, en este caso, Algoritmos de Búsqueda y Ordenamiento, utilizamos para ello el material aportado por la cátedra.
- Elaboración de la introducción y del marco teórico.
- Planteamiento del caso práctico, en este caso un juego desarrollado en Python.
- Ejecución del programa.
- Análisis de los resultados obtenidos mediante la realización de la parte práctica.
- Desarrollo de las conclusiones finales.
- Preparación del anexo:
 - Capturas de pantalla que muestran el funcionamiento del programa creado en Python.
- Armado de carpeta digital en el repositorio en GitHub con el código del programa, el informe del trabajo y la presentación utilizada en el video.
- Armado del video explicativo.

5. Resultados obtenidos

En el desarrollo del juego se implementó de manera exitosa la ejecución de dos algoritmos de búsqueda: búsqueda lineal y búsqueda binaria, y cuatro algoritmos de ordenamiento: Bubble Sort, Selection Sort, Insertion Sort y Quick Sort.

Además, se logró que las funciones que corresponden a cada uno de los algoritmos se ejecuten únicamente cuando la respuesta del usuario fuese correcta, para ello se hizo un uso adecuado de condicionales que fueron controlando la dinámica del quiz.

Aunque el diseño del juego fue sencillo, resultó efectivo para comprender el funcionamiento de estos algoritmos; y la necesidad de generar flujo dinámico para que la ejecución del juego no resultara aburrida para el usuario, agregó cierta complejidad a la propuesta.

Se logró también producir un código modularizado, ya que dividimos el programa en secciones independientes, cada una con su función específica. Esto nos ayudó a que el programa se vea más organizado y limpio. Asimismo, facilitó la división de tareas para el trabajo colaborativo a la hora de la realización del proyecto.

El enfoque que se eligió para la realización del caso práctico facilitó la comprensión tanto teórica como práctica de los algoritmos.

6. Conclusiones

A modo de conclusión, se puede afirmar que este trabajo alcanzó los objetivos que se plantearon en un comienzo.

En relación al primer objetivo, el desarrollo de este trabajo permitió comprender a nivel teórico qué son los algoritmos de búsqueda y de ordenamiento, para que se utilizan, así como también identificar varios de ellos y entender en qué contexto sería mejor su aplicación.

Este aprendizaje teórico fue indispensable para poder aplicar los algoritmos en un caso práctico de forma creativa. La elección de un juego resultó beneficiosa para tener que pensar otras maneras de aplicar los conceptos aprendidos y la forma en la que se diseñó brindó la posibilidad de trabajar colaborativamente de forma sencilla.

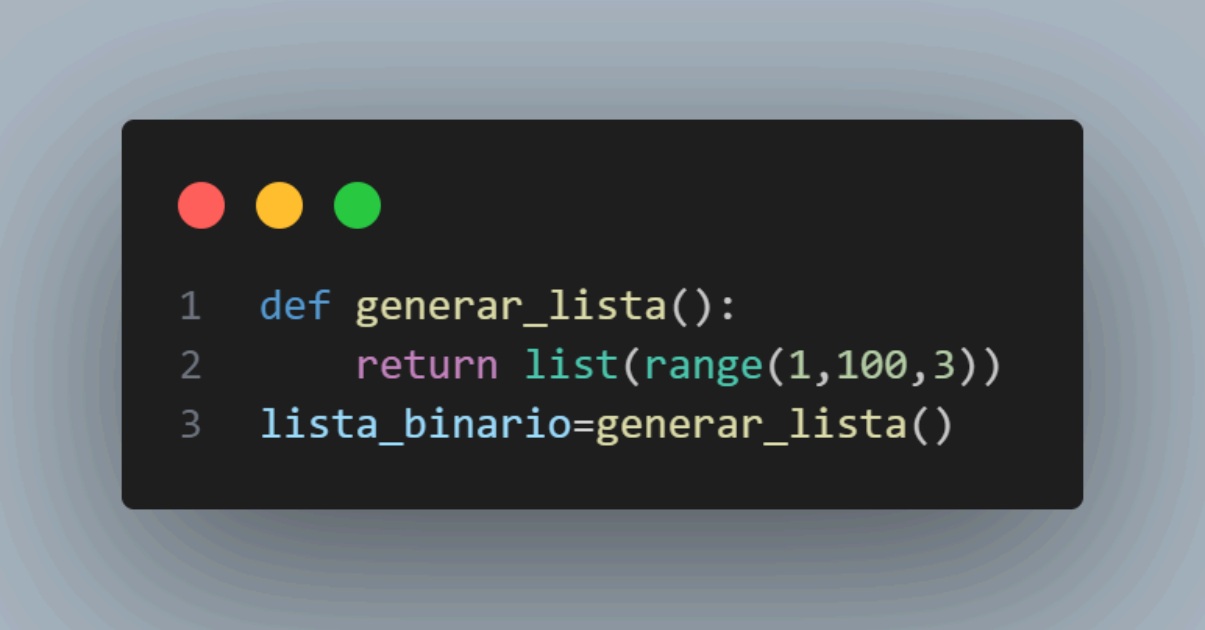
Además, en relación al último objetivo propuesto, se logró entender la utilidad de ambos tipos de algoritmos para los técnicos en programación. Por un lado, porque se encuentran presentes en tareas comunes como buscar elementos en bases de datos y mostrar resultados de forma ordenada. Y por otro lado, porque ofrecen la posibilidad de trabajar con grandes volúmenes de datos y de hacerlo de forma eficiente. Esto requiere de los desarrolladores tener los conocimientos adecuados para poder decidir cual es el algoritmo que se debe utilizar, para así mejorar aún más el rendimiento del programa. Asimismo, la eficiencia que ofrecen no solo está relacionada con el rendimiento, sino que también con el hecho de que ayudan a ahorrar tiempo y recursos.

7. Bibliografía

- *Búsqueda y ordenamiento en programación.* (2025). Material de la cátedra Programación I, Tecnicatura Universitaria en Programación, Universidad Tecnológica Nacional.
- *Clase de programación: Búsquedas y ordenamiento en Python.* (2025). Cátedra Programación I, Tecnicatura Universitaria en Programación, Universidad Tecnológica Nacional. Recuperado de:
<https://colab.research.google.com/drive/1KVqiJSzYLTPDFRwTYjN8CP7G4LP-reD9J?usp=sharing>
- Quirós, N. (2025). Algoritmos de búsqueda. Recuperado de YouTube:
<https://www.youtube.com/watch?v=gJIQTq80llg>
- Quirós, N. (2025). Algoritmos de ordenamiento. Recuperado de YouTube:
<https://www.youtube.com/watch?v=xntUhrhtLaw>

8. Anexos

Se han hecho capturas del código del programa para mayor comprensión, más allá de que se haya puesto a lo largo del trabajo, nos parece importante que también estén acá, las capturas fueron realizadas en Visual Studio Code con la extensión CodeSnap de modo tal que el número de línea de ese código no debe ser tomado en cuenta. En este punto se aplican comillas para dividir el código en algunos casos y así lograr una visualización más clara

A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in Python and is as follows:

```
1  def generar_lista():  
2      return list(range(1,100,3))  
3  lista_binario=generar_lista()
```




```
1 PREGUNTAS = [  
2     # Nivel 1: Búsqueda Lineal  
3     {"nivel": 1,  
4         "texto": ("Si Tienes la siguiente lista: [5, 3, 8, 2, 9, 1]. \n"  
5                     "Debes buscar el número 8 recorriéndola elemento por elemento."  
6                     "¿Qué algoritmo utilizarías?"),  
7         "opciones": ["Búsqueda lineal", "Búsqueda binaria"],  
8         "respuesta_correcta": "1"  
9     },  
10    # Nivel 2: Búsqueda Binaria  
11    {"nivel": 2,  
12        "texto": (f"Si tienes la siguiente lista: {lista_binario} \n"  
13                    "Debes buscar el número 82 en ella, pensando en "  
14                    "la eficiencia de busqueda y la cantidad de valores de la lista."  
15                    " ¿Qué algoritmo utilizarías ?"),  
16        "opciones": ["Búsqueda lineal", "Búsqueda binaria"],  
17        "respuesta_correcta": "2"  
18    },
```



```
1 # Nivel 3: Bubble Sort (Ordenamiento por burbuja)  
2 {"nivel": 3,  
3     "texto": ("Tienes la siguiente lista: [19, 32, 71, 13, 4, 58]. \n"  
4                 "¿Cuál de los siguientes algoritmos usarías para comparar cada "  
5                 " elemento de la lista con el siguiente y luego intercambiarlos "  
6                 " si no están en el orden correcto?"),  
7     "opciones": ["Selection Sort", "Bubble Sort", "Insertion Sort", "Quick Sort"],  
8     "respuesta_correcta": "2"  
9 },  
10  
11 # Nivel 4: Selection Sort (Ordenamiento por selección)  
12 {"nivel": 4,  
13     "texto": ("Tienes la siguiente lista: [66, 23, 17, 4, 33]. \n"  
14                 "¿Qué algoritmo utilizarías para encontrar el elemento menor "  
15                 "de la lista y luego colocarlo en el primer elemento, repitiendo "  
16                 " este proceso con todos los elementos?"),  
17     "opciones": ["Insertion Sort", "Quick Sort", "Selection Sort", "Bubble Sort"],  
18     "respuesta_correcta": "3"  
19 },
```

```

1  # Nivel 5: Insertion Sort (Ordenamiento por inserción)
2  {"nivel": 5,
3   "texto": ("¿Qué algoritmo de ordenamiento construye la lista ordenada elemento"
4             " por elemento, insertando cada nuevo elemento en la posición correcta?"),
5   "opciones": ["Bubble Sort", "Selection Sort", "Insertion Sort", "Quick Sort"],
6   "respuesta_correcta": "3"
7  },
8  # Nivel 6: Quick Sort (Ordenamiento rápido)  ///  Fede
9  {
10   "nivel": 6,
11   "texto": ("¿Cuál de los siguientes algoritmos utiliza el concepto de divide y "
12             "vencerás para ordenar una lista utilizando un pivote?"),
13   "opciones": ["Bubble Sort", "Quick Sort", "Insertion Sort", "Selection Sort"],
14   "respuesta_correcta": "2"
15  },
16 ]

```

```

1  def hacer_pregunta(pregunta):
2      print(f"\nNivel {pregunta['nivel']}:")
3      print(f"{pregunta['texto']}")
4      for i, opcion in enumerate(pregunta['opciones'], 1):
5          print(f"{i}. {opcion}")
6
7      while True:
8          respuesta = input(f"Tu respuesta (número): ")
9          if respuesta.isdigit() and 1 <= int(respuesta) <= len(pregunta['opciones']):
10             break
11          print(f"¡Opción inválida! Ingresas un número entre 1 y {len(pregunta['opciones'])}")
12
13      if respuesta == pregunta['respuesta_correcta']:
14          print("="*80)
15          print(" ¡Correcto!")
16          resolver_algoritmo(pregunta['nivel'])
17          print("="*80)
18          return True
19      else:
20          print(f"Incorrecto. La respuesta correcta era: {pregunta['opciones'][int(pregunta['respuesta_correcta'])-1]}")
21          return False

```



```
1 def busqueda_lineal(lista, objetivo):
2     for i in range(len(lista)):
3         if lista[i] == objetivo:
4             return i
5     return -1
```



```
1 def busqueda_binaria(lista_binario, objetivo):
2     izquierda, derecha = 0, len(lista_binario) - 1
3     while izquierda <= derecha:
4         medio = (izquierda + derecha) // 2
5         if lista_binario[medio] == objetivo:
6             return medio
7         elif lista_binario[medio] < objetivo:
8             izquierda = medio + 1
9         else:
10            derecha = medio - 1
11    return -1
```



```
1 def bubble_sort(lista):
2     for i in range(len(lista)):
3         for j in range(len(lista) - 1 - i):
4             if lista[j] > lista[j + 1]:
5                 lista[j], lista[j + 1] = lista[j + 1], lista[j]
6     return lista
```



```
1 def selection_sort(lista):
2     for i in range(len(lista)):
3         minimo = i
4         for j in range(i + 1, len(lista)):
5             if lista[j] < lista[minimo]:
6                 minimo = j
7         lista[i], lista[minimo] = lista[minimo], lista[i]
8     return lista
```



```
1 def insertion_sort(arr):
2     for i in range(1, len(arr)):
3         key = arr[i]
4         j = i-1
5         while j >= 0 and key < arr[j] :
6             arr[j+1] = arr[j]
7             j -= 1
8         arr[j+1] = key
9     return arr
```



```
1 def quick_sort(arr):
2     if len(arr) <= 1:
3         return arr
4     else:
5         pivot = arr[0]
6         less = [x for x in arr[1:] if x <= pivot]
7         greater = [x for x in arr[1:] if x > pivot]
8         return quick_sort(less) + [pivot] + quick_sort(greater)
```

```

1  def resolver_algoritmo(nivel):
2      # Búsqueda lineal
3      if nivel == 1:
4          lista = [5, 3, 8, 2, 9, 1]
5          print(f"Buscando el número 8 en {lista}" )
6          print(f"El numero se encuentra en la posicion  {busqueda_lineal(lista, 8) + 1} ")
7          print(f"Explicacion: La respuesta corresta es lineal, "
8                "dado que la lista es relativamente corta y es la manera "
9                "de buscar elemento por elemento")
10         # Busqueda Binaria
11     elif nivel == 2:
12         lista = lista_binario
13         print(f"Buscando el número 82 en {lista} ")
14         print(f"Se encuentra en la posicion {busqueda_binaria(lista, 82) + 1} de la lista" )
15         print(f"Explicacion: La respuesta corresta es binaria, "
16               "dado que la lista es bastante grande y se ve claramente que el valor"
17               " requerido esta mas alla de la mitad y este"
18               " es el modo mas eficiente para este tipo de busquedas")
19     # Bubble Sort
20     elif nivel == 3:
21         lista = [19, 32, 71, 13, 4, 58]
22         lista_ordenada = selection_sort(lista.copy())
23         print(f"La lista {lista} ordenada con Bubble sort es {lista_ordenada}")
24         print(f"Explicacion: Bubble Sort compara elementos "
25               "adyacentes y los intercambia si están en el orden"
26               " incorrecto. Repite este proceso hasta que la lista esté "
27               "completamente ordenada.")
28     # Selection_sort
29     elif nivel == 4:
30         lista = [66, 23, 17, 4, 33]
31         lista_ordenada = selection_sort(lista.copy())
32         print(f"La lista {lista} ordenada con Selection es {lista_ordenada}")
33         print(f"Explicación: Selection Sort busca el elemento más pequeño"
34               " en cada iteración y lo coloca en su posición correcta.")
35
36     # Ordenamiento insertion_sort
37     elif nivel == 5:
38         lista = [19, 32, 71, 13, 4, 58]
39         lista_ordenada = insertion_sort(lista.copy())
40         print(f"La lista {lista} ordenada con Isertion es {lista_ordenada}")
41         print(f"Explicación: Insertion Sort construye la lista ordenada un "
42               "elemento a la vez, tomando cada nuevo elemento y colocándolo "
43               "en la posición correcta dentro de la parte ya ordenada.")
44     # Ordenamiento quick_sort
45     elif nivel == 6:
46         lista = [19, 32, 71, 13, 4, 58]
47         lista_ordenada = quick_sort(lista.copy())
48         print(f"La lista {lista} ordenada con Quick sort es {lista_ordenada}")
49         print(f"Explicación: Quick Sort utiliza la técnica 'divide y vencerás':"
50               " selecciona un 'pivote', divide la lista en elementos menores y "
51               "mayores al pivote, y luego repite el proceso recursivamente con cada sublista.")

```

```
1 def jugar():
2     while True:
3         print("="*80)
4         print("BIENVENIDO/A AL QUIZZ DE ALGORITMOS")
5         print("="*80)
6         aciertos = 0
7         for pregunta in PREGUNTAS:
8             if hacer_pregunta(pregunta):
9                 aciertos += 1
10        print("="*80)
11        print(f"Juego terminado. Aciertos: {aciertos}")
12        if aciertos == 6:
13            print(f"¡Perfecto! Has respondido correctamente todas las preguntas.")
14        elif aciertos >= 4:
15            print(f"¡Muy bien! Has respondido bien más de la mitad de las preguntas.")
16        else:
17            print(f"Debes practicar un poco más.")
18
19        volver_a_jugar= input("\n¿Quieres jugar de nuevo? (sí/no): ").lower().strip()
20        if volver_a_jugar != 'si' and volver_a_jugar != 'sí':
21            print("¡Gracias por jugar! ¡Hasta la próxima!")
22            break
```