



Trabajo Práctico Integrador

Algoritmos de ordenamiento y Búsqueda binaria

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Programación 1

Alumnos:

Ignacio Salazar – ignaciosalazarg86@gmail.com

Lautaro Zullo – zullolau@gmail.com

Materia: Programación 1

Profesor: AUS Bruselario, Sebastián

Fecha de Entrega: 09 de junio de 2025

Índice

Índice.....	1
Introducción.....	2
Marco Teórico.....	3
Algoritmos de Búsqueda.....	3
Búsqueda Lineal.....	3
Búsqueda Binaria.....	3
Algoritmos de Ordenamiento.....	5
Bubble sort.....	5
Insertion sort.....	5
Selection sort.....	5
Quicksort.....	5
Tiempos de los algoritmos.....	5
Caso Práctico.....	6
Función del algoritmo Búsqueda Lineal:.....	6
Función del algoritmo Búsqueda Binaria:.....	6
Función del algoritmo Bubble Sort:.....	7
Función del algoritmo QuickSort:.....	7
Metodología Utilizada.....	8
Resultados Obtenidos.....	8
Conclusiones.....	8
Bibliografía.....	9
Anexos.....	9

Introducción

En el presente trabajo práctico integrador abordamos el estudio comparativo de dos algoritmos de ordenamiento: **Bubble Sort** y **Quicksort** y dos algoritmos de búsqueda: **Búsqueda Lineal** y **Búsqueda Binaria**. El objetivo principal fue analizar el rendimiento de cada uno de los métodos mediante la medición de los tiempos de ejecución al ordenar listas de números aleatorios de diferentes tamaños.

Para ello, desarrollamos un algoritmo en Python de forma modular, que genera listas aleatorias y aplica ambos algoritmos de ordenamiento y a ambos algoritmos de búsqueda, registrando el tiempo que tarda cada uno en completarse. El experimento se repite múltiples veces, incrementando progresivamente el tamaño de las listas, hasta detectar en qué punto **Quicksort** comienza a superar en eficiencia a Bubble Sort y en el caso de los algoritmos de búsqueda hasta detectar el punto en el que **Búsqueda Binaria** supere en eficiencia a **Búsqueda Lineal**.

Este trabajo busca no solo comparar el comportamiento práctico de estos algoritmos, sino también reforzar los conceptos teóricos de eficiencia y su aplicación en problemas concretos de programación.

Marco Teórico

Algoritmos de Búsqueda

En computación, un algoritmo de búsqueda es una técnica utilizada para localizar un elemento específico dentro de una estructura de datos, como una lista, un vector o una matriz. Su objetivo principal es encontrar la posición del elemento deseado, o determinar que no se encuentra presente en la estructura. La eficiencia de estos algoritmos depende del tipo de datos, el tamaño de la estructura y si los elementos están ordenados o no.

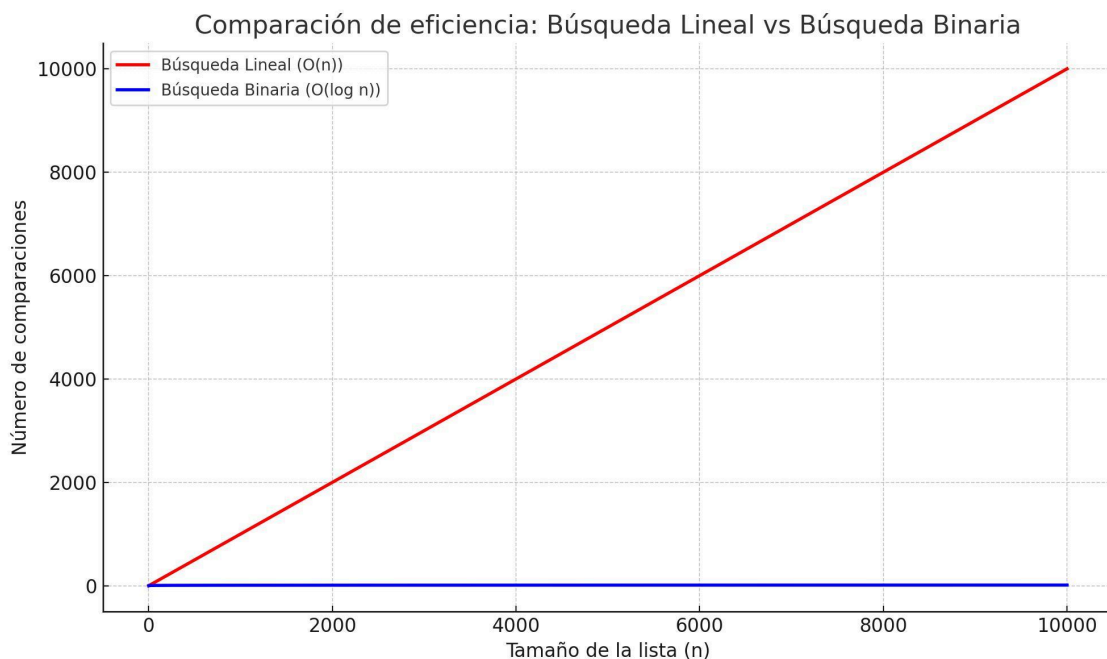
Entre los algoritmos de búsqueda más conocidos se encuentran:

Búsqueda Lineal

La búsqueda lineal, también conocida como búsqueda secuencial, recorre la estructura de datos elemento por elemento desde el principio hasta el final. Compara cada elemento con el valor buscado hasta encontrar una coincidencia o llegar al final sin éxito. Es un algoritmo sencillo de implementar, pero puede resultar ineficiente para estructuras de gran tamaño, ya que en el peor de los casos revisa todos los elementos.

Búsqueda Binaria

La búsqueda binaria es una técnica más eficiente que la búsqueda lineal, pero requiere que los datos estén previamente ordenados. El algoritmo compara el valor buscado con el elemento en el centro de la lista. Si son iguales, se ha encontrado el elemento. Si el valor buscado es menor, se repite el proceso en la mitad izquierda; si es mayor, en la mitad derecha. Este enfoque divide el espacio de búsqueda en cada paso, reduciendo significativamente la cantidad de comparaciones necesarias. La búsqueda binaria se basa en la estrategia de divide y vencerás.



La búsqueda lineal aumenta de manera proporcional al tamaño de la lista: si

hay 1.000 elementos, podría necesitar hasta 1.000 comparaciones en el peor caso.

La búsqueda binaria, en cambio, requiere muchas menos comparaciones: por ejemplo, para 1.000 elementos, sólo necesita alrededor de $\log_2(1000) = 10$ comparaciones.

Algoritmos de Ordenamiento

En computación y matemáticas un algoritmo de ordenamiento nos permite poner elementos de una lista o un vector en una secuencia dada por una relación de orden, es decir, el resultado de salida ha de ser una permutación o reordenamiento de la entrada que satisfaga la relación de orden dada.

Algunos de los algoritmos de ordenamiento más conocidos son:

Bubble sort

Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada

Insertion sort

El Insertion Sort o Ordenamiento por Inserción, divide el conjunto de elementos en una parte ordenada y otra desordenada. Toma un elemento de la parte desordenada y lo inserta en la posición correcta en la parte ordenada. Repite este proceso hasta que todos los elementos estén ordenados.

Selection sort

Selection Sort o Ordenamiento por Selección, busca el elemento más pequeño en el conjunto de elementos y lo coloca en la posición correcta. Luego, busca el siguiente elemento más pequeño y lo coloca en la siguiente posición correcta. Repite este proceso hasta que todos los elementos estén ordenados.

Quicksort

Quicksort o Ordenamiento Rápido, elige un elemento llamado "pivote" y divide el conjunto en dos subconjuntos, uno con elementos menores que el pivote y otro con elementos mayores. Luego, aplica el mismo proceso de forma recursiva en cada uno de los subconjuntos. Este algoritmo también utiliza la estrategia de divide y vencerás.

Tiempos de los algoritmos

Algoritmo	Mejor caso	Peor caso	Caso promedio	Eficiencia	Uso Recomendado
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$	Baja	Listas pequeñas o casi ordenadas
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	Baja	Listas pequeñas
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	Media	Listas pequeñas o casi ordenadas
Quicksort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	Alta	Listas grandes (evitar peor caso)

Caso Práctico

Con el objetivo de observar en la práctica el comportamiento de los algoritmos de Búsqueda y Ordenamiento, se implementó un programa en Python que compara el tiempo de ejecución de los algoritmos de búsqueda lineal y búsqueda binaria y también los algoritmos de ordenamiento Bubble Sort y Quicksort sobre listas de números aleatorios. La lógica consiste en generar listas de tamaño creciente y aplicar ambos algoritmos para medir cuánto tiempo tardan en ordenar la misma lista. El proceso se repite hasta encontrar el punto en el que Búsqueda lineal supera a la búsqueda Binaria y que Quicksort supera a Bubble Sort en eficiencia. A continuación, se presenta el código de las dos pares de funciones utilizadas en la comparación de eficiencia para llevar a cabo esta experimentación:

Función del algoritmo Búsqueda Lineal:

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

Función del algoritmo Búsqueda Binaria:

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

Función del algoritmo Bubble Sort:

```
def bubble_sort(arr):  
    for n in range(len(arr) - 1, 0, -1):  
        swapped = False  
        for i in range(n):  
            if arr[i] > arr[i + 1]:  
                arr[i], arr[i + 1] = arr[i + 1], arr[i]  
                swapped = True  
        if not swapped:  
            break
```

Función del algoritmo QuickSort:

```
def quicksort(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 quicksort(less) + [pivot] + quicksort(greater)
```


Metodología Utilizada

La elaboración del trabajo se realizó en las siguientes etapas:

- Metodología Utilizada
- Recolección de información teórica en documentación confiable.
- Implementación en Python de los algoritmos estudiados.
- Pruebas con diferentes conjuntos de datos.
- Registro de resultados y validación de funcionalidad.
- Elaboración de este informe y preparación de anexos.

Resultados Obtenidos

- El programa realiza búsquedas y ordena correctamente las listas dadas.
- El promedio de elementos en una lista donde quicksort supera a bubble sort es de entre 23 y 27 elementos generalmente.
- Se comprendió el uso y diferencia de usos entre los algoritmos vistos.

Conclusiones

Este trabajo práctico permitió analizar y comparar distintos algoritmos de ordenamiento y búsqueda, observando cómo varía su rendimiento según el tamaño y características de los datos.

En ordenamiento, comprobamos que Bubble Sort, aunque simple de implementar, resulta ineficiente frente a Quicksort, especialmente en listas grandes, donde este último demuestra una clara superioridad.

En cuanto a la búsqueda, la búsqueda lineal, si bien fácil de aplicar, se vuelve poco eficiente en estructuras grandes. Por el contrario, la búsqueda binaria reduce notablemente el número de comparaciones, siempre que los datos estén ordenados.

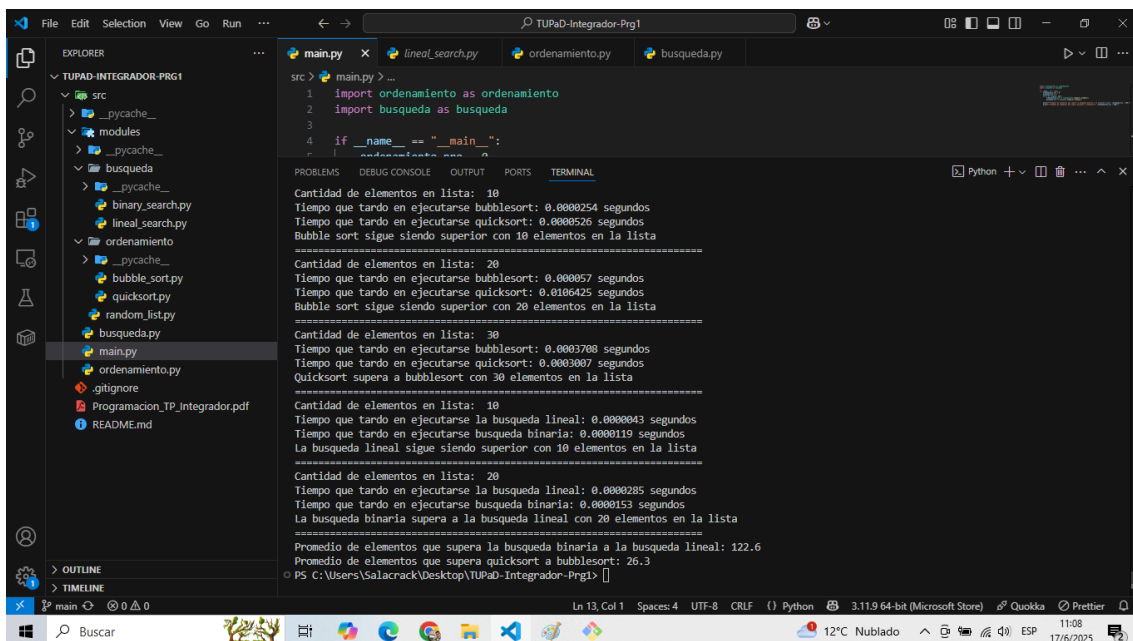
Estas comparaciones refuerzan la importancia de elegir el algoritmo adecuado según el contexto, ya que la eficiencia puede variar significativamente.

Bibliografía

- https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento
- <https://www.geeksforgeeks.org/python-program-for-bubble-sort/>
- <https://www.geeksforgeeks.org/python-program-for-quicksort/>
- <https://docs.python.org/3/library/>
- https://es.wikipedia.org/wiki/Ordenamiento_de_burbuja
- <https://www.swhosting.com/es/comunidad/manual/algoritmos-de-ordenacion-con-ejemplos-en-c#:~:text=El%20Insertion%20Sort%20o%20Ordenamiento,todos%20los%20elementos%20est%C3%A9n%20ordenados.>
- Apuntes en formato pdf utilizados a lo largo de la materia.

Anexos

- Link a repositorio del proyecto:
<https://github.com/LautiZ/TUPaD-Integrador-Prg1>
- <https://youtu.be/TnTu1QWaWco?si=r4tQtFzosK87NxNk>
- Video explicativo: <https://youtu.be/22iWswgVnIU>



```
src > main.py > ...
1 import ordenamiento as ordenamiento
2 import busqueda as busqueda
3
4 if __name__ == "__main__":
    ...

PROBLEMS  DEBUG CONSOLE  OUTPUT  PORTS  TERMINAL
Python + v ...

Cantidad de elementos en lista: 10
Tiempo que tardo en ejecutarse bubblesort: 0.000254 segundos
Tiempo que tardo en ejecutarse quicksort: 0.000526 segundos
Bubble sort sigue siendo superior con 10 elementos en la lista
=====
Cantidad de elementos en lista: 20
Tiempo que tardo en ejecutarse bubblesort: 0.00057 segundos
Tiempo que tardo en ejecutarse quicksort: 0.0106425 segundos
Bubble sort sigue siendo superior con 20 elementos en la lista
=====
Cantidad de elementos en lista: 30
Tiempo que tardo en ejecutarse bubblesort: 0.000378 segundos
Tiempo que tardo en ejecutarse quicksort: 0.0003007 segundos
Quicksort supera a bubblesort con 30 elementos en la lista
=====
Cantidad de elementos en lista: 10
Tiempo que tardo en ejecutarse la busqueda lineal: 0.000043 segundos
Tiempo que tardo en ejecutarse busqueda binaria: 0.000119 segundos
La busqueda lineal sigue siendo superior con 10 elementos en la lista
=====
Cantidad de elementos en lista: 20
Tiempo que tardo en ejecutarse la busqueda lineal: 0.0000285 segundos
Tiempo que tardo en ejecutarse busqueda binaria: 0.000153 segundos
La busqueda binaria supera a la busqueda lineal con 20 elementos en la lista
=====
Promedio de elementos que supera la busqueda binaria a la busqueda lineal: 122.6
Promedio de elementos que supera quicksort a bubblesort: 26.3
PS C:\Users\Salacrack\Desktop\TUPaD-Integrador-Prg1>
```