[](mailto:vcmanuelbarroeta@gmail.com)

Trabajo Práctico Integrador – Implementación y Análisis Comparativo de los Algoritmos de Bubble Sort

Alumnos: Barroeta Victor Manuel Email: [vcmanuelbarroeta@gmail.com](mailto:vcmanuelbarroeta@gmail.com)

Beauvallet Pablo Nahuel Email: [pablonahuelbeauvallet@gmail.com](mailto:pablonahuelbeauvallet@gmail.com)

Materia: Programación I

Profesor: Cinthia Rigoni

Docente tutor: Brian Lara

Fecha de Entrega: 09 de junio de 2025

Índice

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

# INTRODUCCIÓN

En el ámbito de la programación, la eficiencia en la gestión de la información es fundamental para el desarrollo de sistemas que impactan positivamente en la experiencia del usuario y el rendimiento general. Diariamente nosotros utilizamos aplicaciones que dependen críticamente de la búsqueda y el ordenamiento de datos, desde redes sociales, plataformas de streaming hasta motores de búsqueda como Google, donde la rapidez es los más importante en la expectativa central del usuario. Los algoritmos de ordenación que es lo que nos vamos a centrar en este trabajo permiten organizar los elementos de una lista de alguna manera. Podría ser de manera alfabética, ascendente, descendente, etc.

Este trabajo se centrará en entender el funcionamiento de los algoritmos de ordenamiento, centrándose específicamente en la implementación y análisis del algoritmo de burbuja o Bubble Sort. El objetivo es comprender sus limitaciones de rendimiento, ya que es crucial para un programador; tambien sus características operativas, sus ventajas, desventajas, la comparación en otros algoritmos de ordenamiento y su impacto en la organización de colecciones de datos.

Como caso práctico, aplicaremos el Bubble Sort para organizar productos de una compra de supermercado en una lista según su prioridad, específicamente de mayor a menor necesidad. Además de implementar Bubble Sort, se lo comparará con el algoritmo Quick Sort, lo que permitirá apreciar sus diferencias en eficiencia y comportamiento frente a diferentes volúmenes de datos.

Este ejemplo práctico nos permitirá visualizar cómo un algoritmo de ordenamiento simple reestructura una colección de datos para facilitar su interpretación y uso, también nos mostrará cómo comparándolo con otros algoritmos de ordenamiento cuando le añadimos más componentes a la lista. Esto permite tomar decisiones informadas al momento de seleccionar un algoritmo adecuado para resolver problemas reales, evitando crear software lento o ineficiente.

# MARCO TEÓRICO

### ¿Qué es un algoritmo de ordenamiento?

Un algoritmo de ordenamiento es un conjunto de instrucciones o un procedimiento lógico diseñado para reorganizar los elementos de una colección de datos (comúnmente una lista o un arreglo) de acuerdo con un criterio de orden específico. Este criterio puede ser numérico (de menor a mayor, o viceversa), alfabético (de la A a la Z, o viceversa), o cualquier otra relación de orden definida sobre los elementos. Como lo expresa Sánchez (2022), "Un algoritmo de ordenamiento es un procedimiento que reordena los elementos de una lista siguiendo un criterio específico (por ejemplo, de menor a mayor) para facilitar búsquedas y análisis posteriores".

### Importancia:

La relevancia de los algoritmos de ordenamiento en la computación es considerable. Una vez que un conjunto de datos ha sido ordenado, se simplifican y agilizan numerosas operaciones subsecuentes. Primordialmente, facilitan el análisis de datos y son un prerrequisito para la aplicación eficiente de ciertos algoritmos de búsqueda, como la búsqueda binaria, que requiere que la colección de datos esté previamente ordenada para su correcta implementación. La Universidad Tecnológica Nacional (UTN, 2025) subraya que "la importancia de los algoritmos de ordenamiento radica en que posterior al ordenamiento facilitan el trabajo del análisis de datos y los algoritmos de búsqueda". Los beneficios se extienden a una "búsqueda más eficiente", un "análisis de datos más fácil" y la ejecución de "operaciones más rápidas" sobre los datos.

### ¿Qué es el algoritmo Bubble Sort (Ordenamiento burbuja)?

El algoritmo Bubble Sort, conocido como el Ordenamiento de Burbuja, es uno de los métodos más simples de ordenamiento. Aunque es una introducción útil al concepto de algoritmos de ordenamiento para principiantes, su simplicidad viene acompañada de desventajas significativas en términos de eficiencia. Este algoritmo funciona al comparar pares de elementos adyacentes y cambiarlos si están en el orden incorrecto, repitiendo el proceso hasta que la lista esté ordenada. (Celis, 2020).

### ¿Cómo funciona el Bubble Sort?

Ricardo Celis de Platzi afirma que “El Bubble Sort recibe su nombre porque los números más grandes parecen "subir" como burbujas a la parte superior de la lista.” Veamos su funcionamiento:

* **Comparaciones y Cambios**: Recorre la lista repetidamente. En cada recorrido, compara cada par de elementos adyacentes. Si un elemento es mayor que el siguiente, los intercambia.
* **Iteraciones**: Continúa iterando a través de la lista hasta que no se necesiten más cambios, lo que indica que la lista está ordenada.

### Ejemplos de pseudocódigos:

Enseñamos un ejemplo de pseudocódigo mostrado en el material de estudio de la UTN:

Texto

El contenido generado por IA puede ser incorrecto.

### Análisis de Complejidad:

Según el material de estudio de la UTN:

* **Mejor caso:** O(n). Esto ocurre cuando la lista ya está ordenada y se utiliza la versión optimizada. En este escenario, el algoritmo realiza una única pasada para confirmar que no se necesitan intercambios.
* **Peor Caso y caso promedio**: O (n2). Se presenta cuando la lista está ordenada en sentido inverso al deseado (por ejemplo, de mayor a menor si se busca ordenar de menor a mayor). En esta situación, cada elemento debe "viajar" la máxima distancia posible, requiriendo el máximo número de comparaciones e intercambios. Para una lista con elementos en orden aleatorio, el rendimiento se promedió se aproxima al del peor caso.
* **Complejidad Espacial**: O(1). El algoritmo de burbuja es un ordenamiento *in-situ*, ya que solo requiere una cantidad constante de espacio adicional para variables temporales (como la usada para el intercambio).

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

### ¿Por qué es útil aprender Bubble Sort?

Según Ricardo Celis (Platzi, 2020), “A pesar de sus limitaciones, el Bubble Sort es un excelente punto de partida para quienes se inician en el mundo de los algoritmos por su simplicidad. Además, proporciona una base para entender conceptos más avanzados y eficientes, como el Quick Sort o el Merge Sort.”

Sus ventajas son:

* **Fácil comprensión**: La lógica simple de Bubble Sort lo hace ideal para principiantes.
* **Primer paso hacia algoritmos más complejos**: Ayuda a entender los fundamentos de comparaciones y cambios, preparándote para aprender algoritmos más sofisticados.
* **Ejercicios prácticos**: Experimentar con Bubble Sort en el aula o en proyectos de autoaprendizaje puede reforzar destrezas clave de programación.
* Útil para listas pequeñas o ya casi ordenadas
* **Mínimo Uso de Memoria Adicional:** Al ser *in-situ* (O(1) en complejidad espacial), no consume memoria extra significativa, lo cual puede ser una ventaja en entornos con memoria muy limitada.

En cuanto a sus desventajas, (Freitas, s.f.):

* **Ineficiencia para Grandes Conjuntos de Datos**: Su complejidad temporal cuadrática (O(n2)) lo hace extremadamente lento e impráctico para ordenar listas de gran tamaño. Grabiel Freitas (s.f.) advierte: "No es adecuado para grandes conjuntos de datos...".
* **Alto Número de Comparaciones e Intercambios:** Generalmente, realiza un número considerablemente mayor de operaciones de comparación e intercambio en comparación con algoritmos más eficientes, incluso para listas de tamaño moderado.

### ¿Qué es el algoritmo Quick sort (Ordenamiento rapido)?

El algoritmo de ordenamiento rápido, o Quick Sort, es un algoritmo de ordenamiento eficiente y ampliamente utilizado que se basa en la estrategia de "divide y vencerás" (Celis,2020).

### ¿Cómo funciona el Quick Sort?

Su funcionamiento general implica los siguientes pasos recursivos:

* **Selección del Pivote:** Se elige un elemento de la lista, denominado "pivote". La elección del pivote es crucial para la eficiencia del algoritmo. Existen diversas estrategias para seleccionarlo, como tomar el primer elemento, el último, uno aleatorio, o la mediana de tres elementos.
* **Partición:** Una vez seleccionado el pivote, la lista se reorganiza (o particiona) de tal manera que todos los elementos menores que el pivote se colocan a su izquierda, y todos los elementos mayores que el pivote se colocan a su derecha. Los elementos iguales al pivote pueden ir a cualquier lado, o agruparse con el pivote. Al final de este paso, el pivote se encuentra en su posición final ordenada dentro de la lista.
* **Recursión:** El algoritmo se aplica de forma recursiva a las dos sublistas (la de elementos menores que el pivote y la de elementos mayores que el pivote) creadas en el paso de partición. Este proceso continúa hasta que las sublistas contienen cero o un elemento, momento en el cual se consideran ordenadas.

### Ejemplos de pseudocódigos:

Texto

El contenido generado por IA puede ser incorrecto. Enseñamos un ejemplo de pseudocódigo mostrado en el material de estudio de la UTN:

### Análisis de Complejidad:

Según el material de estudio de la UTN:

* **Mejor caso y caso promedio**: O(n log n). Ocurre cuando el pivote elegido en cada paso divide la lista en dos sublistas de tamaño aproximadamente igual, logrando particiones balanceadas. En caso de promedio, ocurre con entradas aleatorias y con una buena estrategia de selección de pivote, tiene este rendimiento y es muy eficiente en la práctica.
* **Peor Caso**: O(n²). Se presenta cuando el pivote elegido es consistentemente el elemento más pequeño o grande de la sublista. Esto lleva a particiones muy desbalanceadas, donde una sublista está vacía y la otra contiene casi todos los elementos, degenerando el comportamiento del algoritmo al de uno cuadrático.
* **Complejidad Espacial:** O (log n). en promedio, ya que utiliza memoria en la pila por ser un algoritmo recursivo. Esa memoria se usa para guardar cada llamada mientras divide la lista en partes más pequeñas. En el peor de los casos, si las divisiones son muy desbalanceadas (por ejemplo, si siempre se elige un mal pivote), puede llegar a O(n), porque la recursión se vuelve muy profunda y consume más memoria. Esto se conoce como una profundidad de recursión lineal.

### ¿Por qué es útil aprender Quick Sort?

Según Ricardo Celis (Platzi, 2020), “El Quick Sort es uno de los algoritmos más eficientes y utilizados en programación. Comprender su funcionamiento ayuda a optimizar el rendimiento de programas y a profundizar en el paradigma de divide y vencerás, esencial en informática avanzada”.

Sus **ventajas** son:

* **Alto rendimiento promedio**: Gracias a su complejidad promedio de O(n log n), Quick Sort es uno de los algoritmos más rápidos para ordenar grandes volúmenes de datos.
* **Aplicación en entornos reales**: Es el algoritmo base de muchas bibliotecas estándar de lenguajes como Java, Python y C++, lo que demuestra su relevancia profesional.
* **Refuerza el pensamiento recursivo:** Implementarlo permite practicar la recursividad, un pilar fundamental en algoritmos y estructuras de datos.
* **Divide y vencerás**: Introduce este patrón de diseño algorítmico, muy útil en muchos problemas complejos.
* **Eficiencia en memoria**: Aunque usa recursividad, no requiere estructuras auxiliares extensas, lo que lo hace relativamente liviano en uso de espacio.

En cuanto a sus **desventajas**:

* **Peor caso poco favorable**: Si el pivote se elige mal (por ejemplo, siempre el primer o último elemento en listas ya ordenadas), puede llegar a tener complejidad O(n²).
* **No es estable**: Puede alterar el orden relativo de elementos con claves iguales, lo cual es una limitación en ciertos contextos donde esto importa.
* **Mayor dificultad de implementación**: Comparado con algoritmos más simples como Bubble o Insertion Sort, requiere mayor atención a la lógica de particionamiento y manejo de índices, lo que puede dificultar su comprensión inicial.

### Análisis Comparativo Teórico: Bubble Sort vs. Quick Sort

En aplicaciones reales donde se manejan grandes volúmenes de datos, casi nunca se elige Bubble Sort, ya que Quick Sort es ampliamente superior en términos de eficiencia promedio. Sin embargo, compararlos es muy útil desde lo teórico y educativo, porque permite entender cómo varía el rendimiento entre diferentes algoritmos de ordenamiento (Freitas (s.f.).

* **Rendimiento general:** Quick Sort es mucho más rápido que Bubble Sort en la mayoría de los casos, especialmente cuando aumenta la cantidad de datos. Esto se debe a sus complejidades promedio: **O(n log n)** para Quick Sort frente a **O(n²)** para Bubble Sort.

**Estabilidad:**

* **Bubble Sort:** Es un algoritmo **estable**, lo que significa que no cambia el orden relativo de elementos iguales.
* **Quick Sort:** No es estable, a menos que se lo modifique, lo que complica su implementación.

**Casos de uso:**

* **Bubble Sort:** Solo se recomienda con fines educativos o para listas pequeñas y casi ordenadas, por su simpleza.
* **Quick Sort:** Es ideal para ordenar listas grandes de manera rápida y es ampliamente utilizado en bibliotecas de programación modernas.

# CASO PRÁCTICO

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto. En esta sección se desarrolló un programa en Python para ordenar una lista de productos de compras futuras según su prioridad (de menor a mayor) se detalla la aplicación práctica de los algoritmos de ordenamiento Bubble Sort y Quick Sort, incluyendo una comparación de sus tiempos de ejecución una lista de productos más amplia.

El código se construyó de la siguiente forma:

En cuanto al desarrollo técnico, se programaron ambos algoritmos en el lenguaje Python. Se comenzó implementando Bubble Sort, validando su funcionamiento con listas simples y observando cómo realizaba las comparaciones e intercambios.

Posteriormente, se incorporó la implementación de Quick Sort y se adaptó el código para realizar pruebas de rendimiento utilizando la biblioteca time, midiendo la eficiencia temporal en listas generadas de forma aleatoria.

Para verificar los resultados, se realizaron múltiples ejecuciones con diferentes volúmenes de datos. Esto permitió identificar claramente las diferencias de rendimiento entre ambos algoritmos, sobre todo en listas grandes, donde Quick Sort demostró ser ampliamente superior.

Finalmente, se desarrolló una gráfica comparativa conceptual para complementar el análisis y facilitar la comprensión visual de las diferencias entre ambos métodos.

**Texto

El contenido generado por IA puede ser incorrecto.**

**Texto

El contenido generado por IA puede ser incorrecto.**

**4- METODOLOGÍA APLICADA**

Para el desarrollo de este trabajo, en principio se planteó una metodología en varias etapas, combinando el estudio conceptual con la simulación práctica de escenarios. A continuación, se describen los pasos seguidos:

* **Investigación previa**

En una primera instancia, se realizó una revisión bibliográfica de diversas fuentes académicas y divulgativas sobre algoritmos de ordenamiento. Se consultaron materiales provistos por la UTN, artículos de en la web, blogs especializados en programación y recursos educativos de Platzi. Esta etapa permitió comprender las características principales de Bubble Sort y Quick Sort, así como sus ventajas, desventajas, complejidades y escenarios de aplicación.

A partir de esta investigación se definió como objetivo la implementación de Bubble Sort y su posterior comparación con Quick Sort, buscando evaluar su comportamiento frente a listas de distintos tamaños, ordenamientos y tiempos de ejecución.

* **Etapas de diseño y prueba del código**

En cuanto al desarrollo técnico, se programaron ambos algoritmos en el lenguaje Python. Se comenzó implementando Bubble Sort, validando su funcionamiento con listas simples y observando cómo realizaba las comparaciones e intercambios.

Posteriormente, se incorporó la implementación de Quick Sort y se adaptó el código para realizar pruebas de rendimiento utilizando la biblioteca time, midiendo la eficiencia temporal en listas generadas de forma aleatoria.

Para verificar los resultados, se realizaron múltiples ejecuciones con diferentes volúmenes de datos. Esto permitió identificar claramente las diferencias de rendimiento entre ambos algoritmos, sobre todo en listas grandes, donde Quick Sort demostró ser ampliamente superior.

Finalmente, se desarrolló una gráfica comparativa conceptual para complementar el análisis y facilitar la comprensión visual de las diferencias entre ambos métodos.

* **Trabajo colaborativo**

El trabajo colaborativo tuvo como eje la prueba y testeo de dos diferentes algoritmos de ordenamiento, aprendiendo y enriqueciéndonos sobre la marcha a través de intercambio que se fueron dando tanto como en esta etapa así también como en posterior crear y modificación del código en etapas en este trabajo práctico.

La repartición de las tareas del grupo se realizó un análisis de los tres temas planteados y se eligió el de búsqueda y ordenamiento, eligiendo el tema de bubble sort comprado con el quick sort. En la parte del marco teórico y la introducción el encargado fue Victor Barroeta. En el caso práctico, metodología y resultados en un inicio fueron realizados por Pablo Beauvallet, teniendo correcciones o cambio mientras se iba desarrollando el trabajo ya que un principio pensamos mostrar nada más el funcionamiento del algoritmo de bubble sort pero después se añadió la comparación con el quick sort de las cual se encargó Victor Barroeta, los resultados que se obtuvieron y se reunió toda la información para realizar el video y la presentación.

**5- RESULTADOS OBTENIDOS**

Al ejecutar el programa, se logró ordenar correctamente la lista de productos según la prioridad asignada, tal como se esperaba. El algoritmo Bubble Sort organizó los elementos desde los de mayor prioridad (valor más bajo) hasta los de menor prioridad, realizando las comparaciones e intercambios necesarios hasta alcanzar el orden deseado.

En una primera prueba con una lista de 6 productos, ambos algoritmos, Bubble Sort y Quick Sort, devolvieron la misma lista ordenada, validando que el funcionamiento lógico de ambos es correcto para ese conjunto de datos.

Texto

El contenido generado por IA puede ser incorrecto.

Luego se implementó una segunda prueba con una lista más extensa de 1000 productos generados aleatoriamente para comparar el tiempo de ejecución de cada algoritmo, después comparamos con 5000 productos y con 10000 productos. Los resultados obtenidos fueron los siguientes:

* **Bubble Sort**: mostró un tiempo de procesamiento notablemente más alto, lo que confirma su ineficiencia para grandes volúmenes de datos.
* **Quick Sort**: completó la tarea de ordenamiento en un tiempo mucho menor, reafirmando su ventaja en rendimiento para listas más largas.

Esta comparación práctica permitió comprobar que, aunque ambos algoritmos funcionan correctamente, Quick Sort es mucho más eficiente en escenarios reales donde el tamaño de los datos importa, mientras que Bubble Sort puede ser útil para listas pequeñas o como herramienta educativa.

También se valoró el uso de tiempo en código real mediante la función time.time(), que permitió medir el rendimiento de forma precisa. Esto reforzó la importancia de considerar la eficiencia en tiempo y memoria al seleccionar un algoritmo en proyectos más complejos.

Enlace a repositorio si el trabajo está subido a GitHub: <https://github.com/victormanbar/TPI_Programacion1_Ordemamiento>

**6- CONCLUSIONES**

Entendimos la importancia del hecho de organizar y estructurar los datos de una manera eficiente ya que esto deriva en que al encontrarse ordenados los datos se puedan realizar búsquedas, análisis y otras operaciones de una manera más rápida y sencilla.

Primero trabajamos con el algoritmo Bubble Sort, que, si bien es sencillo y útil para aprender, no es muy eficiente cuando la lista de datos es grande. Lo usamos para ordenar una lista de productos por prioridad y eso nos ayudó a entender bien cómo funciona.

Después, para comparar, aplicamos Quick Sort, un algoritmo más avanzado y mucho más rápido. Esto nos permitió ver en la práctica que para grandes cantidades de datos es mejor usar algoritmos más eficientes, como Quick Sort, porque tardan mucho menos tiempo en ordenar.

Tenemos que decir, también, que mientras el ordenamiento burbuja es útil para pequeñas listas, en conjuntos de datos grandes es recomendable explorar métodos más eficientes como el ordenamiento rápido, que tiene una complejidad menor y que por lo cual acelera el proceso de organización.

Además de aprender a programar los algoritmos, este trabajo también nos ayudó a practicar cómo comparar distintas soluciones, cómo organizarnos en equipo y cómo documentar lo que hacemos, que son cosas muy importantes para seguir creciendo como programadores.

**7- BIBLIOGRAFÍA**

* UTN (2025). Material de estudio - Búsqueda y ordenamiento. Tecnicatura Universitaria en Programación a Distancia.
* Leira Sánchez (18 de septiembre de 2023). *Algoritmos de Ordenación (Sorting) en JavaScript.* Recuperado de <https://www.escuelafrontend.com/algoritmos-de-ordenacion-javascript>
* Ricardo Celis (10 de marzo de 2020). *Curso de Introducción a los Algoritmos de Ordenamiento.* Recuperado dehttps://platzi.com/cursos/ordenamiento/
* Gabriel Freitas (s.f.). *Algoritmos de ordenación.*Recuperado dehttps://www.studysmarter.es/resumenes/ciencias-de-la-computacion/algoritmos-en-ciencias-de-la-computacion/algoritmos-de-ordenacion/