INGENIER ÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN

Docente

ALFONSO RECIO HERNANDEZ

EVIDENCIAS DE LA MATERIA:

ESTRUCTURA Y ORGANIZACION DE DATOS TC1 2025 A

Reporte de Algoritmos de ordenamiento y búsqueda

Participante:
Castro Limón Gustavo Eduardo

Número de control: 23151227

Semestre: 04

Semestre: enero-mayo de 2025

Aguascalientes, Ags. 25 de Mayo de 2025

1. ¿Qué son los algoritmos de ordenamiento?

Son técnicas fundamentales en la informática que permiten organizar una colección de elementos en un orden específico, ya sea ascendente o descendente. Estos algoritmos son esenciales en diversas aplicaciones, desde la búsqueda de datos hasta la optimización de procesos.

Características de los algoritmos de ordenamiento (complejidades estabilidad, adaptabilidad, simplicidad, entre otras)

Bubble Sort
Es un algoritmo simple que compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Este proceso se repite hasta que la lista está ordenada. Es lento para listas grandes, pero fácil de entender e ideal para enseñar conceptos básicos.

Selection Sort
Este algoritmo recorre la lista y selecciona el elemento más pequeño (o más grande) y lo coloca en su posición final, repitiendo el proceso con el resto de la lista. No es muy eficiente, pero no necesita memoria adicional.

Insertion Sort
Inserta cada elemento en su lugar correcto dentro de una lista parcialmente ordenada, desplazando los elementos mayores hacia la derecha. Funciona muy bien en listas pequeñas o casi ordenadas.

Merge Sort
Divide la lista en mitades recursivamente hasta tener listas de un solo elemento y luego las mezcla ordenadamente. Es muy eficiente y estable, pero necesita memoria adicional.

Heap Sort
Convierte la lista en una estructura de heap (montículo), luego extrae repetidamente el elemento máximo (o mínimo) para construir la lista ordenada. Es eficiente y no requiere memoria adicional, pero no es estable.

Quick Sort
Elige un "pivote" y divide la lista en dos partes: menores y mayores que el pivote. Luego ordena recursivamente ambas partes. Muy rápido en la práctica, pero su peor caso puede ser lento si el pivote no se elige bien.

3-Way Quick Sort
Una variante del Quick Sort que divide la lista en tres partes: menores, iguales y mayores que el pivote. Ideal para listas con muchos elementos duplicados, mejora el rendimiento en esos casos.

Algoritmo	Complejidad	Estabilidad	Adaptabilidad	Simplicidad	Detalles
Bubble Sort	$O(n^2)$	Sí	Sí	Alta	Fácil de implementar, útil en educación.

Algoritmo	Complejidad	Estabilidad	Adaptabilidad	Simplicidad	Detalles
Selection Sort	$O(n^2)$	No	No	Alta	No requiere memoria adicional, pero no es eficiente.
Insertion Sort	$O(n^2)$ (mejor caso $O(n)$)	Sí	Sí	Alta	Muy eficiente para listas pequeñas o casi ordenadas.
Merge Sort	$O(n \log n)$	Sí	No	Media	Divide y conquista, necesita memoria adicional.
Heap Sort	$O(n \log n)$	No	No	Media	Eficiente y sin uso extra de memoria, pero inestable.
Quick Sort	$O(n \log n)$ promedio, $O(n^2)$ peor	No	No	Media	Muy rápido en la práctica, pero no estable.
3-Way Quick Sort	$O(n \log n)$ (mejor con duplicados)	No	Sí	Media	Ideal para listas con muchos elementos repetidos.

Términos:
Complejidad: Tiempo que tarda en el peor o promedio de los casos (O-grande).
Estabilidad: Mantiene un relativo orden de los elementos con claves iguales.
Adaptabilidad: Se comporta mejor si los datos están parcialmente ordenados.
Simplicidad: Facilidad de implementación y comprensión.

Aplicaciones de los algoritmos de ordenamiento

Búsqueda y Recuperación de Datos: Ordenar datos facilita su búsqueda eficiente, por ejemplo, en sistemas de bases de datos y motores de búsqueda.

Optimización de Procesos: Ayuda a optimizar procesos en sistemas operativos, como la planificación de tareas.

Análisis de Datos: Herramienta fundamental en el análisis de datos para ordenar y visualizar información de manera coherente.

Algoritmos y Estructuras de Datos: Muchos algoritmos avanzados y estructuras de datos, como los árboles de búsqueda y las colas de prioridad, dependen de técnicas de ordenamiento.

Ordenamiento por Burbuja (Bubble Sort)
Usos: Educación. Listas pequeñas.

Ordenamiento por Selección (Selection Sort)
Usos: Sistemas con memoria limitada. Listas pequeñas.

Ordenamiento por Inserción (Insertion Sort)
Usos: Listas pequeñas o casi ordenadas. Ordenamiento en tiempo real.

Ordenamiento por Mezcla (Merge Sort)
Usos: Grandes conjuntos de datos. Ordenamiento externo (en disco).

Ordenamiento Apilado (Heap Sort)
Usos: Sistemas operativos (planificación de tareas). Colas de prioridad.

Ordenamiento Rápido (Quick Sort)
Usos: Ordenamiento en memoria principal. Bases de datos y librerías estándar.

Ordenamiento en 3 Caminos (3-Way QuickSort)
Usos: Listas con muchos duplicados. Procesamiento paralelo de datos.

Código:

Bubble sort: import random

import time

Tamaño del arreglo

tam = 1000000

lim_inf = 1

lim_sup = 100000

Generar arreglo aleatorio

inicio_gen = time.time()

arreglo_original = [random.randint(lim_inf, lim_sup) for _ in range(tam)]

fin_gen = time.time()

print(*f*"Tiempo para generar arreglo: {fin_gen - inicio_gen*:.6f*} segundos")

Copiar el arreglo para ordenarlo

arreglo = arreglo_original[:]

Bubble Sort

inicio_sort = time.time()

```
n = len(arreglo)
for i in range(n):
   for j in range(0, n-i-1):
      if arreglo[j] > arreglo[j+1]:
         arreglo[j], arreglo[j+1] = arreglo[j+1], arreglo[j]
fin_sort = time.time()

# Resultados
print(f"\nBubble:")
print(f"Tiempo de ordenamiento de columna: {fin_sort - inicio_sort:.6f} segundos")
print(f"Últimos 100 valores ordenados:\n{arreglo[-100:]}")
```

Insertion sort:

```
import random
import time
# Tamaño del arreglo
tam = 100000
lim_inf = 1
lim_sup = 100000
# Generar arreglo aleatorio
inicio_gen = time.time()
arreglo_original = [random.randint(lim_inf, lim_sup) for _ in range(tam)]
fin_gen = time.time()
print(f"Tiempo para generar arreglo: {fin_gen - inicio_gen:.6f} segundos")

# Copiar el arreglo para ordenarlo
arreglo = arreglo_original[:]
# Insertion Sort
inicio_sort = time.time()
n = len(arreglo)
```

```
for i in range(1, n):
    clave = arreglo[i]
    j = i - 1
    while j >= 0 and arreglo[j] > clave:
        arreglo[j + 1] = arreglo[j]
        j -= 1
    arreglo[j + 1] = clave
fin_sort = time.time()

# Resultados
print(f"\nInsertion:")
print(f"Tiempo de ordenamiento de columna: {fin_sort - inicio_sort:.6f} segundos")
print(f"Últimos 100 valores ordenados:\n{arreglo[-100:]}")
```

Quick sort:

```
import random
import time
# Tamaño del arreglo
tam = 1000000
lim_inf = 1
lim_sup = 100000
# Generar arreglo aleatorio
inicio_gen = time.time()
arreglo_original = [random.randint(lim_inf, lim_sup) for _ in range(tam)]
fin_gen = time.time()
print(f"Tiempo para generar arreglo: {fin_gen - inicio_gen:.6f} segundos")

# Copiar el arreglo para ordenarlo
arreglo = arreglo_original[:]
# QuickSort
```

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivote = arr[len(arr) // 2]
    menores = [x for x in arr if x < pivote]
    iguales = [x for x in arr if x == pivote]
    mayores = [x for x in arr if x > pivote]
    return quicksort(menores) + iguales + quicksort(mayores)

inicio_sort = time.time()
arreglo = quicksort(arreglo)
fin_sort = time.time()

# Resultados
print(f"\nMétodo Rápido (QuickSort):")
print(f"Tiempo de ordenamiento de columna: {fin_sort - inicio_sort:.6f} segundos")
print(f"Últimos 100 valores ordenados:\n{arreglo[-100:]}")
```

Selection sort: import random

```
import time
# Tamaño del arreglo
tam = 100000
lim_inf = 1
lim_sup = 100000
# Generar arreglo aleatorio
inicio_gen = time.time()
arreglo_original = [random.randint(lim_inf, lim_sup) for _ in range(tam)]
fin_gen = time.time()
print(f"Tiempo para generar arreglo: {fin_gen - inicio_gen:.6f} segundos")
```

```
# Copiar el arreglo para ordenarlo
arreglo = arreglo_original[:]
# Selection Sort
inicio_sort = time.time()
n = len(arreglo)
for i in range(n):
    min_idx = i
    for j in range(i+1, n):
        if arreglo[j] < arreglo[min_idx]:
            min_idx = j
    arreglo[i], arreglo[min_idx] = arreglo[min_idx], arreglo[i]
fin_sort = time.time()

# Resultados
print(f"\nSelection:")
print(f"Tiempo de ordenamiento de columna: {fin_sort - inicio_sort:.6f} segundos")
print(f"Últimos 100 valores ordenados:\n{arreglo[-100:]}")
```

Pruebas de algoritmos de ordenamiento 1

	BubbleSort
1,000	C:\Users\maryf\Downloads>py BubbleSort.py Tiempo para generar arreglo: 0.000998 segundos Bubble: Tiempo de ordenamiento de columna: 0.086481 segundos Últimos 100 valores ordenados: [90927, 90934, 90941, 91101, 91108, 91160, 91195, 91399, 91416, 91637, 91757, 91936, 92114, 92115, 92128, 92165, 92186, 92309, 92317, 92433, 92481, 92629, 92831, 92931, 92938, 92979, 93054, 93081, 93162, 93172, 93328, 93481, 93727, 93801, 93919, 94121, 94278, 94406, 94436, 94460, 94465, 94540, 94668, 94671, 94799, 94865, 94865, 94871, 94920, 95033, 95039, 95169, 95470, 95506, 95540, 95551, 95608, 95707, 96323, 96348, 96526, 96564, 96637, 96724, 96851, 96877, 97126, 97156, 97178, 97220, 97344, 97457, 97471, 97567, 97646, 97677, 97784, 97804, 97812, 98066, 98146, 98161, 98211, 98428, 98469, 98650, 98790, 98821, 98831, 98874, 99257, 99392, 99448, 99541, 99604, 99727, 99810, 99814, 99931, 99986]

10,000	C:\Users\maryf\Downloads>py BubbleSort.py Tiempo para generar arreglo: 0.003788 segundos Bubble: Tiempo de ordenamiento de columna: 30.503098 segundos Últimos 100 valores ordenados: [99049, 99060, 99062, 99062, 99065, 99069, 99084, 99113, 99114, 99126, 99130, 99135, 99152, 99156, 99161, 99169, 99198, 99208, 99215, 99217, 99218, 99226, 99235, 99237, 99282, 99287, 99298, 99304, 99321, 99325, 99338, 99345, 99355, 99365, 99372, 99390, 99398, 99400, 99409, 99419, 99425, 99426, 99437, 99442, 99453, 99456, 99461, 99470, 99472, 99472, 99477, 99482, 99492, 99494, 99501, 99529, 99539, 99551, 99555, 99557, 99564, 99574, 99576, 99582, 99590, 99593, 99598, 99607, 99626, 99647, 99649, 99650, 99661, 99678, 99680, 99685, 99685, 99702, 99714, 99722, 99724, 99729, 99729, 99751, 99767, 99783, 99794, 99813, 99820, 99856, 99891, 99910, 99917, 99922, 99945, 99960, 99966, 99981, 99986, 99995]
100,000	C:\Users\maryf\Downloads>py BubbleSort.py Tiempo para generar arreglo: 0.003788 segundos Bubble: Tiempo de ordenamiento de columna: 30.503098 segundos Últimos 100 valores ordenados: [99049, 99060, 99062, 99062, 99065, 99069, 99084, 99113, 99114, 99126, 99130, 99135, 99152, 99156, 99161, 99169, 99198, 99208, 99215, 99217, 99218, 99226, 99235, 99237, 99282, 99287, 99298, 99304, 99321, 99325, 99338, 99345, 99355, 99365, 99372, 99390, 99398, 99400, 99409, 99419, 99425, 99426, 99437, 99442, 99453, 99456, 99461, 99470, 99472, 99472, 99477, 99482, 99492, 99494, 99501, 99529, 99539, 99551, 99555, 99557, 99564, 99574, 99576, 99582, 99590, 99593, 99598, 99607, 99626, 99647, 99649, 99650, 99661, 99678, 99680, 99685, 99685, 99702, 99714, 99722, 99724, 99729, 99729, 99751, 99767, 99783, 99794, 99813, 99820, 99856, 99891, 99910, 99917, 99922, 99945, 99960, 99966, 99981, 99986, 99995]

InsertionSort	
1,000	C:\Users\maryf\Downloads>py InsertionSort.py Tiempo para generar arreglo: 0.000698 segundos Insertion: Tiempo de ordenamiento de columna: 0.062340 segundos Últimos 100 valores ordenados: [90939, 90966, 91072, 91139, 91234, 91257, 91509, 91524, 91537, 91546, 91603, 91855, 91855, 91993, 92516, 92689, 92741, 92938, 93081, 93253, 93273, 93278, 93443, 93631, 93670, 93772, 93812, 93997, 94153, 94228, 94274, 94312, 94323, 94490, 94505, 94591, 94697, 94755, 94918, 95044, 95273, 95286, 95320, 95431, 95685, 95728, 95751, 95975, 96030, 96048, 96210, 96228, 96282, 96367, 96455, 96498, 96558, 96566, 96577, 96589, 96717, 96788, 96854, 96924, 96936, 96963, 96982, 96984, 97029, 97068, 97175, 97336, 97571, 97624, 97893, 97921, 97973, 97987, 98072, 98507, 98516, 98558, 98559, 98606, 98665, 98706, 98793, 98794, 99024, 99123, 99138, 99195, 99322, 99360, 99453, 99462, 99588, 99826, 99918, 99967]
10,000	C:\Users\maryf\Downloads>py InsertionSort.py Tiempo para generar arreglo: 0.006352 segundos Insertion: Tiempo de ordenamiento de columna: 7.348141 segundos Últimos 100 valores ordenados: [98973, 98989, 98991, 99016, 99021, 99024, 99035, 99036, 99042, 99045, 99076, 99085, 99090, 99094, 99103, 99105, 99106, 99107, 99109, 99119, 99144, 99158, 99192, 99217, 99232, 99239, 99247, 99254, 99294, 99299, 99306, 99310, 99323, 99326, 99336, 99344, 99358, 99390, 99395, 99395, 99412, 99425, 99436, 99444, 99482, 99499, 99499, 99518, 99523, 99531, 99536, 99544, 99556, 99565, 99572, 99573, 99575, 99604, 99607, 99648, 99648, 99658, 99660, 99665, 99667, 99687, 99697, 99706, 99707, 99713, 99758, 99768, 99771, 99777, 99781, 99786, 99790, 99808, 99820, 99820, 99824, 99851, 99851, 99855, 99863, 99865, 99876, 99887, 99894, 99896, 99900, 99913, 99915, 99930, 99932, 99936, 99942, 99947, 99971, 99986]
100,000	C:\Users\maryf\Downloads>py InsertionSort.py Tiempo para generar arreglo: 0.049293 segundos Insertion: Tiempo de ordenamiento de columna: 1681.542317 segundos Últimos 100 valores ordenados: [99900, 99902, 99902, 99904, 99904, 99905, 99907, 99909, 99910, 99911, 99912, 99915, 99916, 99917, 99918, 99919, 99920, 99921, 99921, 99922, 99923, 99923, 99924, 99924, 99929, 99930, 99930, 99932, 99933, 99933, 99933, 99933, 99936, 99937, 99937, 99937, 99938, 99939, 99941, 99942, 99943, 99944, 99945, 99945, 99946, 99952, 99953, 99955, 99956, 99956, 99958, 99958, 99961, 99963, 99966, 99966, 99967, 99967, 99968, 99968, 99968, 99969, 99969, 99969, 99971, 99972, 99972, 99972, 99973, 99973, 99975, 99976, 99977, 99978, 99979, 99980, 99982, 99983, 99983, 99983, 99984, 99984, 99984, 99986, 99986, 99987, 99988, 99989, 99990, 99990, 99990, 99991, 99992, 99992, 99993, 99994, 99995, 99996, 99998, 99999]

QuickSort

1,000	C:\Users\maryf\Downloads>py QuickSort.py Tiempo para generar arreglo: 0.001363 segundos Método Rápido (QuickSort): Tiempo de ordenamiento de columna: 0.002741 segundos Últimos 100 valores ordenados: [90810, 90902, 91075, 91095, 91422, 91432, 91540, 91674, 91734, 91751, 91754, 91861, 91904, 92158, 92247, 92451, 92464, 92494, 92684, 92732, 92766, 92831, 92854, 92879, 92944, 92958, 93078, 93239, 93325, 93643, 93658, 93753, 93824, 93895, 93997, 94111, 94157, 94332, 94344, 94534, 94560, 94638, 94674, 94727, 94753, 94946, 94973, 94994, 95001, 95045, 95108, 95175, 95224, 95411, 95644, 95645, 95690, 95749, 96015, 96150, 96312, 96411, 96422, 96424, 96553, 96580, 96581, 96667, 96759, 96919, 96989, 97006, 97115, 97249, 97512, 97722, 97770, 98019, 98139, 98283, 98376, 98533, 98623, 98658, 98669, 98703, 98805, 98853, 98889, 98957, 98969, 99033, 99093, 99324, 99493, 99514, 99724, 99797, 99927, 99951]
10,000	PS C:\Users\maryf\Downloads> py .\QuickSort.py Tiempo para generar arreglo: 0.006260 segundos Método Rápido (QuickSort): Tiempo de ordenamiento de columna: 0.016486 segundos Últimos 100 valores ordenados: [98995, 99005, 99024, 99064, 99067, 99076, 99091, 99091, 99098, 99099, 99107, 99146, 99161, 99171, 99182, 99185, 99217, 99217, 99243, 99251, 99279, 99296, 99311, 99315, 99324, 99334, 99343, 99363, 99370, 99380, 99390, 99401, 99411, 99413, 99416, 99416, 99416, 99423, 99441, 99442, 99450, 99460, 99469, 99495, 99503, 99511, 99512, 99513, 99553, 99568, 99569, 99596, 99631, 99644, 99663, 99668, 99673, 99674, 99686, 99702, 99705, 99710, 99711, 99719, 99724, 99744, 99760, 99764, 99778, 99779, 99792, 99793, 99795, 99811, 99812, 99824, 99830, 99832, 99832, 99832, 99835, 99846, 99849, 99861, 99863, 99896, 99901, 99909, 99925, 99935, 99937, 99941, 99954, 99966, 99969, 99975, 99981, 99982, 99985, 99986]
100,000	PS C:\Users\maryf\Downloads> py .\QuickSort.py Tiempo para generar arreglo: 0.039190 segundos Método Rápido (QuickSort): Tiempo de ordenamiento de columna: 0.184503 segundos Últimos 100 valores ordenados: [99904, 99905, 99906, 99906, 99907, 99908, 99909, 99909, 99910, 99912, 99914, 99915, 99916, 99916, 99916, 99916, 99917, 99918, 99918, 99919, 99919, 99919, 99920, 99921, 99926, 99926, 99927, 99929, 99929, 99932, 99934, 99935, 99936, 99936, 99936, 99938, 99938, 99940, 99941, 99941, 99941, 99942, 99942, 99943, 99944, 99944, 99946, 99946, 99948, 99949, 99951, 99951, 99951, 99952, 99952, 99954, 99955, 99955, 99957, 99957, 99958, 99959, 99962, 99963, 99964, 99964, 99964, 99965, 99965, 99966, 99967, 99967, 99969, 99969, 99971, 99971, 99971, 99972, 99974, 99977, 99978, 99979, 99980, 99980, 99980, 99981, 99983, 99984, 99984, 99985, 99988, 99988, 99988, 99990, 99993, 99994, 99994, 99997, 99998, 100000]
1M	PS C:\Users\maryf\Downloads> py .\QuickSort.py Tiempo para generar arreglo: 0.426829 segundos Método Rápido (QuickSort): Tiempo de ordenamiento de columna: 2.356535 segundos Últimos 100 valores ordenados: [99990, 99990, 99990, 99990, 99990, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99993, 99993, 99993, 99993, 99993, 99993, 99993, 99993, 99993, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99995, 99995, 99995, 99995, 99995, 99995, 99995, 99995, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000]

SelectionSort	
1,000	C:\Users\maryf\Downloads>py SelectionSort.py Tiempo para generar arreglo: 0.001308 segundos Selection: Tiempo de ordenamiento de columna: 0.100265 segundos Últimos 100 valores ordenados: [92539, 92555, 92612, 92646, 92691, 92727, 92738, 92769, 92820, 92912, 92942, 92984, 93066, 93078, 93101, 93141, 93176, 93346, 93348, 93349, 93368, 93443, 93493, 93602, 93669, 93798, 93838, 93920, 93943, 94258, 94408, 94515, 94545, 94567, 94569, 94701, 94728, 94852, 94958, 95219, 95266, 95326, 95368, 95471, 95561, 95683, 95714, 95721, 95759, 95938, 96001, 96043, 96165, 96201, 96277, 96556, 96721, 96759, 96774, 96778, 96832, 96889, 96970, 97029, 97055, 97117, 97146, 97172, 97252, 97280, 97290, 97331, 97335, 97427, 97481, 97565, 97802, 97964, 98056, 98098, 98236, 98242, 98422, 98451, 98500, 98543, 98554, 98800, 98856, 98889, 98917, 98939, 99284, 99295, 99513, 99534, 99646, 99796, 99837, 99857]
10,000	C:\Users\maryf\Downloads>py SelectionSort.py Tiempo para generar arreglo: 0.015266 segundos Selection: Tiempo de ordenamiento de columna: 14.491059 segundos Últimos 100 valores ordenados: [98904, 98905, 98905, 98906, 98908, 98918, 98932, 98940, 98947, 98960, 98962, 98976, 98987, 99007, 99008, 99016, 99018, 99039, 99042, 99046, 99047, 99064, 99064, 99074, 99084, 99085, 99099, 99123, 99148, 99161, 99165, 99174, 99197, 99215, 99233, 99234, 99240, 99240, 99248, 99249, 99259, 99285, 99289, 99289, 99299, 99310, 99320, 99322, 99329, 99333, 99347, 99360, 99362, 99380, 99405, 99406, 99430, 99431, 99435, 99443, 99450, 99451, 99468, 99486, 99510, 99518, 99545, 99545, 99591, 99622, 99655, 99658, 99661, 99666, 99674, 99676, 99688, 99709, 99774, 99789, 99794, 99798, 99799, 99816, 99841, 99844, 99853, 99854, 99854, 99856, 99898, 99930, 99937, 99943, 99968, 99971, 99981, 99985, 99988, 99991]

100,000	C:\Users\maryf\Downloads>py SelectionSort.py Tiempo para generar arreglo: 0.094871 segundos Selection: Tiempo de ordenamiento de columna: 1675.897266 segundos Últimos 100 valores ordenados: [99902, 99903, 99904, 99904, 99907, 99910, 99910, 99912, 99914, 99920, 99921, 99921, 99922, 99922, 99922, 99923, 99923, 99924, 99924, 99925, 99926, 99926, 99927, 99928, 99929, 99930, 99930, 99930, 99931, 99931, 99932, 99936, 99936, 99937, 99939, 99939, 99939, 99940, 99944, 99944, 99944, 99945, 99948, 99950, 99953, 99954, 99954, 99955, 99955, 99955, 99956, 99956, 99957, 99958, 99960, 99961, 99961, 99962, 99962, 99963, 99963, 99963, 99963, 99964, 99965, 99966, 99966, 99967, 99967, 99968, 99969, 99969, 99970, 99971, 99972, 99972, 99974, 99975, 99976, 99978, 99983, 99983, 99984, 99984, 99985, 99987, 99988, 99989, 99990, 99990, 99991, 99992, 99992, 99994, 99996, 99998, 99998, 100000, 100000, 100000]

Pruebas de algoritmos de ordenamiento 2

BubbleSort	
1,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/BubbleSort.py" Tiempo para generar arreglo: 0.001047 segundos Bubble: Tiempo de ordenamiento de columna: 0.229547 segundos Últimos 100 valores ordenados: [90817, 91115, 91124, 91165, 91188, 91288, 91397, 91398, 91449, 91521, 91556, 91593, 91629, 91657, 91779, 91871, 91873, 91900, 91942, 92354, 92422, 92598, 92691, 92764, 92767, 92822, 93043, 93063, 93214, 93300, 93482, 93598, 93687, 93689, 93809, 93917, 93985, 94142, 94152, 94183, 94210, 94294, 94368, 94515, 94809, 94980, 94981, 95113, 95161, 95241, 95270, 95389, 95681, 95715, 95820, 95932, 95947, 96202, 96246, 96255, 96275, 96308, 96334, 96566, 96611, 96981, 97103, 97112, 97225, 97318, 97435, 97435, 97500, 97895, 97910, 98064, 98148, 98180, 98353, 98370, 98384, 98384, 98436, 98483, 98493, 98505, 98914, 98944, 98954, 99005, 99114, 99160, 99185, 99240, 99595, 99727, 99805, 99838, 99959, 99997]
10,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/BubbleSort.py" Tiempo para generar arreglo: 0.000831 segundos Bubble: Tiempo de ordenamiento de columna: 0.354243 segundos Últimos 100 valores ordenados: [89499, 89808, 89852, 89935, 89979, 90016, 90029, 90065, 90536, 90609, 90683, 90867, 90955, 91067, 91104, 91136, 91480, 91559, 91575, 91608, 91668, 91709, 91746, 91799, 91825, 91933, 92039, 92361, 92454, 92457, 92463, 92713, 92945, 93200, 93207, 93323, 93433, 93516, 93589, 93675, 93794, 93836, 94498, 94898, 94905, 94949, 95106, 95208, 95212, 95214, 95216, 95357, 95437, 95625, 95628, 95774, 95962, 96042, 96203, 96238, 96400, 96407, 96791, 96800, 97070, 97142, 97168, 97406, 97513, 97520, 97547, 97670, 97687, 97808, 97914, 97968, 98006, 98009, 98245, 98270, 98285, 98693, 98706, 98719, 98881, 98891, 98935, 99053, 99075, 99135, 99183, 99216, 99418, 99431, 99467, 99472, 99524, 99802, 99862, 99887]
100,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/BubbleSort.py" Tiempo para generar arreglo: 0.009588 segundos Bubble: Tiempo de ordenamiento de columna: 21.226433 segundos Últimos 100 valores ordenados: [98832, 98853, 98861, 98862, 98875, 98877, 98883, 98895, 98899, 98914, 98921, 98936, 98948, 98971, 98972, 98979, 98980, 98985, 98990, 99017, 99049, 99089, 99089, 99092, 99104, 99107, 99111, 99117, 99129, 99142, 99146, 99147, 99150, 99160, 99181, 99198, 99211, 99223, 99251, 99254, 99271, 99300, 99333, 99363, 99368, 99377, 99385, 99395, 99398, 99408, 99433, 99442, 99474, 99475, 99478, 99484, 99511, 99513, 99513, 99574, 99577, 99579, 99583, 99594, 99601, 99602, 99622, 99625, 99635, 99639, 99639, 99657, 99679, 99696, 99696, 99699, 99708, 99735, 99747, 99747, 99758, 99765, 99776, 99809, 99810, 99813, 99815, 99822, 99828, 99830, 99871, 99886, 99904, 99909, 99910, 99918, 99919, 99945, 99949, 99968]
1M	Tiempo para generar arreglo: 1.100550 segundos Bubble: Tiempo de ordenamiento de columna: 131502.740000 segundos Últimos 100 valores ordenados: [99989, 99989, 99989, 99989, 99990, 99990, 99990, 99990, 99990, 99990, 99990, 99990, 99990, 99990, 99990, 99991, 99991, 99991, 99991, 99991, 99991, 99992, 99992, 99992, 99992, 99993, 99993, 99993, 99993, 99993, 99993, 99993, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99995, 99995, 99995, 99995, 99995, 99995, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000]

InsertionSort

1,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/InsertionSort.py" Tiempo para generar arreglo: 0.000929 segundos Insertion: Tiempo de ordenamiento de columna: 0.119368 segundos Últimos 100 valores ordenados: [90092, 90190, 90282, 90354, 90396, 90397, 90683, 90828, 90900, 90933, 90976, 90999, 91045, 91184, 91233, 91526, 91670, 91978, 91988, 92078, 92094, 92103, 92442, 92458, 92551, 92577, 92600, 93162, 93162, 93211, 93475, 93488, 93672, 93736, 93821, 93880, 93969, 94032, 94151, 94433, 94485, 94651, 94694, 94782, 94822, 94986, 95113, 95165, 95363, 95402, 95405, 95444, 95496, 95506, 95511, 95615, 95928, 96030, 96051, 96082, 96160, 96163, 96218, 96222, 96243, 96261, 96328, 96330, 96412, 96537, 96556, 96758, 97059, 97068, 97378, 97405, 97509, 97624, 97753, 97759, 97777, 97807, 97922, 97946, 98308, 98404, 98523, 98718, 98863, 98970, 98995, 99120, 99233, 99299, 99407, 99625, 99703, 99791, 99930, 99964]
10,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/InsertionSort.py" Tiempo para generar arreglo: 0.009924 segundos Insertion: Tiempo de ordenamiento de columna: 11.361515 segundos Últimos 100 valores ordenados: [99012, 99015, 99030, 99032, 99054, 99073, 99078, 99088, 99112, 99122, 99123, 99128, 99129, 99134, 99144, 99162, 99164, 99165, 99180, 99196, 99208, 99226, 99229, 99232, 99235, 99249, 99250, 99263, 99264, 99270, 99280, 99282, 99286, 99304, 99312, 99315, 99324, 99335, 99342, 99368, 99374, 99385, 99395, 99400, 99405, 99410, 99423, 99430, 99431, 99433, 99437, 99442, 99443, 99523, 99529, 99545, 99552, 99555, 99559, 99570, 99580, 99583, 99588, 99620, 99624, 99634, 99642, 99677, 99682, 99684, 99684, 99704, 99707, 99735, 99747, 99766, 99768, 99774, 99776, 99779, 99795, 99798, 99815, 99816, 99823, 99826, 99832, 99835, 99838, 99847, 99871, 99884, 99895, 99902, 99926, 99943, 99943, 99965, 99982, 99995]
100,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/InsertionSort.py" Tiempo para generar arreglo: 0.097479 segundos Insertion: Tiempo de ordenamiento de columna: 1338.577514 segundos Últimos 100 valores ordenados: [99878, 99878, 99878, 99883, 99883, 99885, 99886, 99886, 99887, 99888, 99888, 99889, 99889, 99889, 99890, 99890, 99890, 99891, 99892, 99894, 99895, 99896, 99898, 99900, 99902, 99902, 99904, 99904, 99907, 99908, 99910, 99911, 99911, 99913, 99914, 99914, 99915, 99918, 99918, 99920, 99921, 99921, 99925, 99925, 99925, 99926, 99928, 99928, 99929, 99930, 99932, 99937, 99937, 99938, 99938, 99940, 99941, 99942, 99943, 99943, 99945, 99945, 99949, 99951, 99952, 99952, 99952, 99956, 99957, 99957, 99959, 99959, 99961, 99961, 99962, 99965, 99967, 99968, 99968, 99969, 99971, 99974, 99975, 99975, 99976, 99977, 99979, 99979, 99981, 99982, 99982, 99983, 99985, 99987, 99988, 99990, 99994, 99996, 99997, 100000]
1M	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/InsertionSort.py" Tiempo para generar arreglo: 1.100834 segundos Insertion: Tiempo de ordenamiento de columna: 82933.820000 segundos Últimos 100 valores ordenados: [99990, 99990, 99990, 99990, 99990, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99993, 99993, 99993, 99993, 99993, 99993, 99993, 99993, 99993, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99995, 99995, 99995, 99995, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000]

QuickSort	
1,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/QuickSort.py" Tiempo para generar arreglo: 0.000923 segundos Método Rápido (QuickSort): Tiempo de ordenamiento de columna: 0.002682 segundos Últimos 100 valores ordenados: [89896, 89916, 90042, 90118, 90213, 90301, 90620, 90639, 90736, 90762, 90809, 90831, 90873, 90874, 91073, 91303, 91400, 91440, 91478, 91532, 91645, 91689, 91722, 91883, 91945, 92095, 92239, 92283, 92339, 92410, 92554, 92706, 92778, 92936, 93065, 93506, 93589, 93670, 93708, 93815, 93911, 93930, 93990, 94007, 94016, 94081, 94438, 94559, 94640, 94643, 94671, 94755, 94814, 94861, 94962, 95113, 95216, 95274, 95277, 95480, 95611, 95749, 96106, 96316, 96380, 96480, 96483, 96921, 96993, 97251, 97517, 97539, 97859, 97876, 97889, 98117, 98155, 98157, 98299, 98448, 98460, 98560, 98652, 98859, 98910, 99144, 99176, 99202, 99214, 99439, 99441, 99443, 99546, 99583, 99704, 99843, 99851, 99858, 99887, 99978]

10,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/QuickSort.py" Tiempo para generar arreglo: 0.010086 segundos Método Rápido (QuickSort): Tiempo de ordenamiento de columna: 0.035715 segundos Últimos 100 valores ordenados: [99180, 99187, 99194, 99197, 99200, 99206, 99217, 99218, 99223, 99236, 99238, 99239, 99242, 99247, 99252, 99256, 99292, 99298, 99300, 99332, 99341, 99358, 99383, 99385, 99396, 99404, 99409, 99416, 99417, 99424, 99425, 99434, 99436, 99440, 99452, 99453, 99454, 99459, 99469, 99472, 99479, 99484, 99490, 99493, 99499, 99548, 99564, 99576, 99578, 99590, 99596, 99607, 99608, 99612, 99619, 99626, 99634, 99647, 99647, 99654, 99660, 99662, 99662, 99670, 99683, 99691, 99721, 99725, 99743, 99771, 99788, 99791, 99792, 99814, 99819, 99831, 99832, 99838, 99849, 99856, 99862, 99866, 99868, 99875, 99879, 99880, 99881, 99890, 99892, 99896, 99896, 99901, 99907, 99915, 99917, 99920, 99940, 99950, 99957, 99979]
100,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/QuickSort.py" Tiempo para generar arreglo: 0.116369 segundos Método Rápido (QuickSort): Tiempo de ordenamiento de columna: 0.590563 segundos Últimos 100 valores ordenados: [99902, 99903, 99903, 99904, 99904, 99906, 99909, 99909, 99910, 99910, 99911, 99912, 99914, 99915, 99916, 99917, 99918, 99918, 99918, 99920, 99923, 99924, 99924, 99924, 99926, 99927, 99927, 99928, 99929, 99929, 99930, 99931, 99931, 99932, 99932, 99936, 99936, 99937, 99939, 99939, 99940, 99941, 99943, 99943, 99943, 99944, 99945, 99946, 99947, 99949, 99950, 99953, 99953, 99954, 99955, 99956, 99957, 99963, 99965, 99966, 99968, 99968, 99969, 99970, 99971, 99972, 99972, 99973, 99973, 99976, 99977, 99977, 99977, 99978, 99978, 99980, 99980, 99981, 99983, 99984, 99984, 99984, 99986, 99987, 99987, 99988, 99989, 99989, 99990, 99990, 99990, 99990, 99995, 99996, 99996, 99998, 99998, 99998, 99999, 100000]
1M	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/QuickSort.py" Tiempo para generar arreglo: 1.122950 segundos Método Rápido (QuickSort): Tiempo de ordenamiento de columna: 8.640616 segundos Últimos 100 valores ordenados: [99990, 99990, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99991, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99992, 99993, 99993, 99993, 99993, 99993, 99993, 99993, 99993, 99993, 99993, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99995, 99995, 99995, 99995, 99995, 99995, 99995, 99995, 99995, 99995, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99997, 99998, 99998, 99998, 99998, 99998, 99998, 99999, 99999, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000]

SelectionSort	
1,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/SelectionSort.py" Tiempo para generar arreglo: 0.001078 segundos Selection: Tiempo de ordenamiento de columna: 0.123371 segundos Últimos 100 valores ordenados: [89214, 89327, 89413, 89436, 89482, 89920, 89931, 90194, 90213, 90301, 90525, 90648, 90669, 90950, 91006, 91259, 91276, 91400, 91523, 91569, 91671, 91697, 91782, 91915, 91995, 91999, 92321, 92374, 92414, 92626, 92645, 92655, 92678, 92747, 92793, 92872, 92937, 93053, 93253, 93399, 93492, 93517, 93537, 93909, 94118, 94257, 94341, 94583, 94638, 94689, 94798, 94828, 95006, 95059, 95315, 95409, 95547, 95619, 95654, 95717, 95856, 95995, 96103, 96159, 96349, 96431, 96497, 96615, 96776, 96922, 97159, 97214, 97474, 97709, 97809, 97888, 97959, 97975, 97975, 97978, 98003, 98259, 98264, 98391, 98506, 98562, 98639, 98732, 98840, 98852, 99024, 99130, 99210, 99370, 99450, 99475, 99568, 99659, 99726, 99902]
10,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/SelectionSort.py" Tiempo para generar arreglo: 0.010079 segundos Selection: Tiempo de ordenamiento de columna: 12.321277 segundos Últimos 100 valores ordenados: [99139, 99140, 99149, 99152, 99157, 99165, 99170, 99197, 99197, 99212, 99229, 99229, 99245, 99255, 99255, 99261, 99266, 99272, 99276, 99290, 99294, 99308, 99309, 99314, 99315, 99321, 99325, 99330, 99340, 99346, 99355, 99359, 99366, 99385, 99385, 99387, 99388, 99393, 99402, 99412, 99412, 99415, 99418, 99424, 99426, 99439, 99467, 99492, 99506, 99511, 99513, 99526, 99539, 99550, 99567, 99569, 99577, 99577, 99578, 99578, 99586, 99616, 99617, 99630, 99636, 99649, 99669, 99692, 99693, 99725, 99737, 99737, 99746, 99773, 99777, 99787, 99797, 99804, 99810, 99811, 99821, 99834, 99857, 99864, 99877, 99880, 99885, 99886, 99901, 99906, 99924, 99929, 99932, 99934, 99938, 99942, 99966, 99979, 99985, 99988]

100,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/SelectionSort.py" Tiempo para generar arreglo: 0.094925 segundos Selection: Tiempo de ordenamiento de columna: 1205.400343 segundos Últimos 100 valores ordenados: [99898, 99898, 99905, 99907, 99907, 99907, 99907, 99908, 99909, 99909, 99910, 99910, 99911, 99911, 99912, 99913, 99913, 99914, 99916, 99916, 99916, 99917, 99917, 99918, 99919, 99924, 99924, 99925, 99926, 99926, 99926, 99928, 99928, 99928, 99929, 99930, 99931, 99931, 99931, 99932, 99932, 99934, 99936, 99937, 99937, 99939, 99940, 99941, 99941, 99941, 99942, 99943, 99944, 99945, 99945, 99946, 99946, 99946, 99947, 99948, 99950, 99951, 99953, 99955, 99955, 99957, 99957, 99958, 99958, 99960, 99960, 99966, 99967, 99968, 99970, 99971, 99972, 99972, 99976, 99976, 99978, 99980, 99984, 99985, 99985, 99986, 99986, 99987, 99987, 99987, 99988, 99991, 99992, 99995, 99997, 99998, 99998, 99999, 99999, 99999]
1M	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/SelectionSort.py" Tiempo para generar arreglo: 1.100358 segundos Selection: Tiempo de ordenamiento de columna: 76310.100000 segundos Últimos 100 valores ordenados: [99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99994, 99995, 99995, 99995, 99995, 99995, 99995, 99995, 99995, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99996, 99997, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99998, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 99999, 100000]

Conclusiones sobre tiempos, estabilidad, cantidad de valores ordenados, comparación con otros equipos de cómputo etc.

Conclusiones:
Tiempo: El método más rápido fue Quicksort mientras que bubble fue el más lento y selection e insertion son normales en promedio, aunque me sorprendió que al menos en mi caso, para el arreglo de 100 mil el de burbuja fue el más rápido, no tengo entendido porque sucede esto, y no creo que el código este mal, de ahí en el método por excelencia es el de quicksort.
Estabilidad: Me asusta el hecho de 1M de números para los de insertion, bubble y selection ya que son muy ineficientes para ello, y aunque si pueden cumplir su objetivo se ve que no están hechos para cantidades enormes.
Cantidad de valores: Obviamente hay una diferencia entre 1000 números a 100000 cada vez va siendo más grande y por eso tarda más en ordenar los números sin embargo, se nota que el de burbuja no esta hecho para cantidades grandes ya que se tarda mucho más que los demás como por 10 minutos más, esto es por su complejidad de o(n a la 1 o 2) eso hace que tarde más en ordenar.
Comparación con otros equipos: La diferencia es mínima, tal vez porque los procesadores y las demás especificaciones son parecidas pero a mi me dio la impresión de que son las mismas, si acaso la diferencia es entre 1 a 4 minutos con cantidades grandes.

2. ¿Qué son los algoritmos de búsqueda?

Un algoritmo de búsqueda es un conjunto de instrucciones que están diseñadas para localizar un elemento con ciertas propiedades dentro de una estructura de datos. Un concepto fundacional de la informática aplicable a campos como los motores de búsqueda, las bases de datos y la inteligencia artificial.

Características de los algoritmos de búsqueda (complejidades estabilidad, adaptabilidad, simplicidad, entre otras)

1. Búsqueda Secuencial

Es el algoritmo más simple de búsqueda. Recorre la lista o arreglo desde el inicio hasta el final, comparando cada elemento con el valor que se busca. No requiere que los datos estén ordenados. Es fácil de implementar, pero ineficiente en listas grandes.
Ventaja: Funciona en cualquier colección.
Desventaja: Es lento si la lista es larga.

2. Búsqueda Binaria

Es mucho más eficiente que la búsqueda lineal, pero solo funciona con listas ordenadas. Compara el valor buscado con el elemento del medio; si no coincide, descarta la mitad que no puede contener el valor, y repite el proceso en la otra mitad.
Ventaja: Muy rápida en listas grandes ordenadas.
Desventaja: Solo sirve si los datos están previamente ordenados.

3. Búsqueda en Profundidad DFS

Es un algoritmo para grafos que explora primero los caminos más profundos antes de retroceder. Puede usarse para recorrer grafos o árboles, resolver laberintos o detectar ciclos. Se implementa con una pila (explícita o usando recursividad).
Ventaja: Útil para explorar soluciones completas o ramas largas.
Desventaja: Puede quedar atrapado en caminos profundos sin solución si no se controla.

4. Búsqueda en Amplitud BFS

Explora todos los nodos vecinos primero antes de ir más profundo. Se implementa usando una cola. Es ideal para encontrar el camino más corto en un grafo sin pesos o para niveles por nivel (como en juegos, redes o inteligencia artificial).
Ventaja: Encuentra el camino más corto en grafos no ponderados.
Desventaja: Puede usar mucha memoria si el grafo es muy grande.

Algoritmo	Complejidad	Estabilidad	Adaptabilidad	Simplicidad	Detalles
Búsqueda Secuencial	O(n)	Sí	Sí	Alta	Recorre la lista elemento por elemento. Funciona con cualquier tipo de estructura, incluso no ordenada.
Búsqueda Binaria	O(log n) (en lista ordenada)	Sí	No	Media	Solo funciona con listas ordenadas. Divide y conquista. Muy rápida en listas grandes ordenadas.

Algoritmo	Complejidad	Estabilidad	Adaptabilidad	Simplicidad	Detalles
Búsqueda en Profundidad (DFS)	O(V + E) (V: vértices, E: aristas)	No	Sí	Media	Explora lo más profundo posible antes de retroceder. Útil en grafos grandes y laberintos. Puede implementarse con recursión o pila.
Búsqueda en Amplitud (BFS)	O(V + E)	No	Sí	Media	Explora nivel por nivel. Usa cola. Ideal para encontrar caminos más cortos en grafos no ponderados. Muy útil en IA y grafos.

Términos:

- **Estabilidad**: En búsquedas, indica si puede distinguir entre elementos repetidos manteniendo orden relativo o priorizando apariciones.
- **Adaptabilidad**: Se comporta mejor si los datos están en cierto orden o estructura (por ejemplo, si ya están ordenados o el grafo es denso).
- **Simplicidad**: Qué tan fácil es de implementar y entender.
- **Complejidad**: Tiempo que tarda en el peor o promedio de los casos (O-grande).

Aplicaciones de los algoritmos de búsqueda

- Encontrar un nombre en una guía telefónica
- Localizar un número en una matriz ordenada
- Hallar una ruta en un laberinto
- Encontrar una palabra clave en un documento
- Búsqueda de datos en grandes matrices ordenadas
- Búsqueda en sistemas de archivos
- Búsqueda de rutas
- Planificación
- Machine Learning

Estos algoritmos de búsqueda se adaptan a diferentes estructuras de datos y pueden requerir un conocimiento previo de los datos. La selección del algoritmo de búsqueda depende de la estructura de los datos y de la familiaridad con los mismos

Código:

```
import random
import time
# Tamaños de los arreglos
tam = 1000000
lim_inf = 1
lim_sup = 100000

# Generar arreglo aleatorio
def generar_arreglo(tam, lim_inf, lim_sup):
    return [random.randint(lim_inf, lim_sup) for _ in range(tam)] #regresa el arreglo

# Algoritmo de ordenamiento QuickSort (in-place)
def particion(arr, inicio, fin):
    pivote = arr[inicio]  # Elegimos el primer elemento como pivote
    low = inicio + 1 # Comenzamos desde el siguiente elemento
    high = fin # Comenzamos desde el último elemento

    while True:
        while low <= high and arr[high] >= pivote: # Incrementamos low hasta encontrar un elemento mayor que el pivote
            high -= 1 # Decrementamos high hasta encontrar un elemento menor que el pivote
        while low <= high and arr[low] <= pivote: # Incrementamos low hasta encontrar un elemento menor que el pivote
            low += 1    # Decrementamos high hasta encontrar un elemento mayor que el pivote
        if low <= high: # Si los índices no se han cruzado
            arr[low], arr[high] = arr[high], arr[low]  # Intercambia si no se han cruzado los índices
        else:
            break # Si los índices se han cruzado, salimos del bucle
```

```
    arr[inicio], arr[high] = arr[high], arr[inicio]  # Coloca el pivote en su lugar final
    return high # Retorna la posición del pivote
# Función recursiva para QuickSort
def quick_sort(arr, inicio, fin): # Ordena el arreglo entre los índices inicio y fin
    if inicio >= fin:
        return
    p = particion(arr, inicio, fin)  # Obtener la posición del pivote
    quick_sort(arr, inicio, p - 1)   # Ordenar parte izquierda
    quick_sort(arr, p + 1, fin)      # Ordenar parte derecha

# Búsqueda secuencial (lineal)
def busqueda_secuencial(arr, valor):
    iteraciones = 0
    for i in range(len(arr)): # Recorre el arreglo
        iteraciones += 1 # Incrementa el contador de iteraciones
        if arr[i] == valor: # Si el elemento actual es igual al valor buscado
            return i, iteraciones # Retorna el índice y el número de iteraciones
    return -1, iteraciones # Si no se encuentra el valor, retorna -1 y el número de iteraciones

# Búsqueda binaria
def busqueda_binaria(arr, valor):
    inicio = 0
    fin = len(arr) - 1 # Inicializa los límites de búsqueda
    iteraciones = 0
    while inicio <= fin:
        iteraciones += 1
        medio = (inicio + fin) // 2 # Calcula el índice medio
        if arr[medio] == valor: # Si el elemento en el medio es igual al valor buscado
            return medio, iteraciones # Retorna el índice medio y el número de iteraciones
```

```
        elif arr[medio] < valor: # Si el elemento en el medio es menor que el valor buscado
            inicio = medio + 1 # Ajusta el inicio para buscar en la mitad derecha
        else: #sino
            fin = medio - 1 # Ajusta el fin para buscar en la mitad izquierda
    return -1, iteraciones

# Generar arreglo aleatorio
inicio_gen = time.time()
arreglo = generar_arreglo(tam, lim_inf, lim_sup)
fin_gen = time.time()

# Ordenar con QuickSort
inicio_sort = time.time()
quick_sort(arreglo, 0, len(arreglo) - 1)
fin_sort = time.time()

# Valor a buscar dentro del arreglo
valor_buscado = arreglo[len(arreglo) // 2]  # Uno que sabemos que existe

# Búsqueda Secuencial
inicio_seq = time.time()
indice_seq, iter_seq = busqueda_secuencial(arreglo, valor_buscado)
fin_seq = time.time()
print(f"\nBúsqueda Secuencial:")
print(f"Valor encontrado en índice: {indice_seq}" if indice_seq != -1 else "Valor no encontrado")
print(f"Iteraciones: {iter_seq}")
print(f"Tiempo de búsqueda: {fin_seq - inicio_seq:.6f} segundos")

# Búsqueda Binaria
```

```
inicio_bin = time.time()

indice_bin, iter_bin = busqueda_binaria(arreglo, valor_buscado)

fin_bin = time.time()

print(f"\nBúsqueda Binaria:")

print(f"Valor encontrado en índice: {indice_bin}" if indice_bin != -1 else "Valor no encontrado")

print(f"Iteraciones: {iter_bin}")

print(f"Tiempo de búsqueda: {fin_bin - inicio_bin:.6f} segundos")
```

Pruebas de algoritmos de búsqueda equipo 1

Secuencial (Quicksort)	
1,000	PS C:\Users\maryf> & C:/Users/maryf/AppData/Local/Programs/Python/Python313/python.exe c:/Users/maryf/Downloads/QuickSearch.py Búsqueda Secuencial: Valor encontrado en índice: 500 Iteraciones: 501 Tiempo de búsqueda: 0.000028 segundos
10,000	PS C:\Users\maryf> & C:/Users/maryf/AppData/Local/Programs/Python/Python313/python.exe c:/Users/maryf/Downloads/QuickSearch.py Búsqueda Secuencial: Valor encontrado en índice: 5000 Iteraciones: 5001 Tiempo de búsqueda: 0.000303 segundos
100,000	PS C:\Users\maryf> & C:/Users/maryf/AppData/Local/Programs/Python/Python313/python.exe c:/Users/maryf/Downloads/QuickSearch.py Búsqueda Secuencial: Valor encontrado en índice: 50000 Iteraciones: 50001 Tiempo de búsqueda: 0.004120 segundos
1M	PS C:\Users\maryf> & C:/Users/maryf/AppData/Local/Programs/Python/Python313/python.exe c:/Users/maryf/Downloads/QuickSearch.py Búsqueda Secuencial: Valor encontrado en índice: 500000 Iteraciones: 500001 Tiempo de búsqueda: 0.083516 segundos

Binaria (Quicksort)	
1,000	Búsqueda Binaria: Valor encontrado en índice: 500 Iteraciones: 9 Tiempo de búsqueda: 0.000011 segundos PS C:\Users\maryf>
10,000	Búsqueda Binaria: Valor encontrado en índice: 5000 Iteraciones: 13 Tiempo de búsqueda: 0.000011 segundos PS C:\Users\maryf>

100,000	Búsqueda Binaria: Valor encontrado en índice: 50000 Iteraciones: 16 Tiempo de búsqueda: 0.000021 segundos PS C:\Users\maryf>
1M	Búsqueda Binaria: Valor encontrado en índice: 500006 Iteraciones: 17 Tiempo de búsqueda: 0.000027 segundos PS C:\Users\maryf>

Pruebas de algoritmos de búsqueda equipo 2

Secuencial	
1,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/QuickSearch.py" Búsqueda Secuencial: Valor encontrado en índice: 500 Iteraciones: 501 Tiempo de búsqueda: 0.000051 segundos
10,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/QuickSearch.py" Búsqueda Secuencial: Valor encontrado en índice: 5000 Iteraciones: 5001 Tiempo de búsqueda: 0.001030 segundos
100,000	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/QuickSearch.py" Búsqueda Secuencial: Valor encontrado en índice: 49999 Iteraciones: 50000 Tiempo de búsqueda: 0.012843 segundos
1M	PS C:\Users\gusta> & C:/Users/gusta/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/gusta/Downloads/estructura recio/QuickSearch.py" Búsqueda Secuencial: Valor encontrado en índice: 499992 Iteraciones: 499993 Tiempo de búsqueda: 0.126923 segundos

Binaria	
1,000	Búsqueda Binaria: Valor encontrado en índice: 500 Iteraciones: 9 Tiempo de búsqueda: 0.000024 segundos PS C:\Users\gusta>

10,000	Búsqueda Binaria: Valor encontrado en índice: 5000 Iteraciones: 13 Tiempo de búsqueda: 0.000027 segundos PS C:\Users\gusta>	
100,000	Búsqueda Binaria: Valor encontrado en índice: 49999 Iteraciones: 1 Tiempo de búsqueda: 0.000015 segundos PS C:\Users\gusta>	
1M	Búsqueda Binaria: Valor encontrado en índice: 499999 Iteraciones: 1 Tiempo de búsqueda: 0.000019 segundos PS C:\Users\gusta>	

Conclusiones sobre tiempos, estabilidad, cantidad de valores, comparación con otros equipos de cómputo etc.

Conclusiones:

Tiempo: La búsqueda secuencial es mucho más lenta en comparación con la búsqueda binaria la binaria le toma muy poco tiempo milésimas de segundos a diferencia que la secuencial casi se acerca a minutos dado esto, binaria por excelencia.

Estabilidad: Ambas búsquedas son lo bastante estables para realizar búsquedas con más de 1 millón de valores dada mis pruebas.

Cantidad de valores: Observo mucho que en la búsqueda secuencial toma demasiadas iteraciones para realizar una búsqueda eficiente y este número incrementa exponencialmente cuando mayor sea la cantidad de datos o números en el arreglo, a diferencia que el binario que le toma pocas o menos de 20 iteraciones y escala progresivamente entre 3 a 5 iteraciones cuando mayor sea el numero de datos o en este caso cuando hay muchos más números en el arreglo.

Comparación con otros equipos: Lo mismo diferencia es mínima, tal vez porque los procesadores y las demás especificaciones son parecidas pero a mí me dio la impresión de que son las mismas, si acaso la diferencia es de segundos con cantidades grandes.

Conclusiones del reporte

No es de esperarse que la mejor forma de implementar el ordenamiento como dicen muchas paginas y expertos en el tema es el de QuickSort y para su posterior búsqueda de algún elemento y poco uso de recursos lo mas recomendable es usar Búsqueda Binaria esto para casos de un numero masivo de datos, para cantidades pequeñas puede utilizar cualquiera de los otros métodos incluso los dados por excelencia en este reporte todo depende como de la estructura de datos y su conocimiento previo de los datos. La selección del algoritmo de ordenamiento y búsqueda depende de la estructura de los datos y de la familiaridad con los mismos

Información del equipo 1 de pruebas

Device name	Desktop-839h7
Processor	12th Gen Intel(R) Core(TM) i5-12500H 3.10 GHz
Installed RAM	16.0 GB (15.7 GB usable)
Device ID	E982F7BA-7EDE-4A39-B150-09C687DFC47F
Product ID	00342-22091-09617-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display
Edition	Windows 11 Home
Version	24H2
Installed on	12/23/2024
OS build	26100.4061
Experience	Windows Feature Experience Pack 1000.26100.84.0

Información del equipo 2 de pruebas

Nombre del dispositivo	DESKTOP-9AKMRPM
Procesador	AMD A12-9720P RADEON R7, 12 COMPUTE CORES 4C+8G 2.70 GHz
RAM instalada	12.0 GB (11.4 GB utilizable)
Almacenamiento	447 GB SSD SAMGET-480GB
Tarjeta gráfica	AMD Radeon R7 Graphics (499 MB)
Identificador de dispositivo	577A4FB5-9048-4031-88C1-8C62F0DADC39
Id. del producto	00325-80837-74378-AAOEM
Tipo de sistema	Sistema operativo de 64 bits, procesador x64

Lápiz y entrada táctil La entrada táctil o manuscrita no está disponible para esta pantalla

Edición Windows 10 Home

Versión 2009

Instalado el 13/01/2025

Compilación del sistema operativo 19045.5737

Referencias bibliográficas

https://www.geeksforgeeks.org/python-program-for-selection-sort/
https://www.geeksforgeeks.org/insertion-sort-algorithm/
https://www.geeksforgeeks.org/quick-sort-algorithm/
https://www.geeksforgeeks.org/bubble-sort-algorithm/
https://www.freecodecamp.org/espanol/news/algoritmos-de-ordenacion-explicados-con-ejemplos-en-javascript-python-java-y-c/
https://architecnologia.es/programacion-c-algoritmos-ordenamiento-busqueda-apuntadores
https://www.geeksforgeeks.org/sorting-algorithms/
https://certidevs.com/tutorial-fundamentos-programacion-algoritmos-ordenamiento
https://www.khanacademy.org/computing/computer-science/algorithms
https://en.wikipedia.org/wiki/Sorting_algorithm#Comparison_of_algorithms
https://inteligenciaonlineblog.wordpress.com/2017/03/04/busqueda-ciega/
https://www.luigisbox.es/blog/tipos-de-algoritmos-de-busqueda/
https://www.geeksforgeeks.org/linear-search/
https://www.geeksforgeeks.org/binary-search/
https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
https://medium.com/@mise/algoritmos-de-b%C3%BAsqueda-y-ordenamiento-7116bcea03d0
https://4geeks.com/es/lesson/algoritmos-de-ordenamiento-y-busqueda-en-python
https://www.pereiratechtalks.com/analisis-de-algoritmos-de-ordenamiento/