Algoritmos de ordenamiento y su comparación

Gael Alpízar Alfaro C20270

***Resumen*— El trabajo consiste en implementar varios algoritmos de ordenamiento en C++ y analizar su eficiencia. Entre estos algoritmos se encuentran el de ordenamiento por selección, inserción y mezcla. Se realizarán experimentos para recolectar información sobre el rendimiento de estos algoritmos, ejecutándolos varias veces con diferentes tamaños de arreglos y capturando el tiempo de ejecución en milisegundos. Se analizará la variación de los tiempos en tres ejecuciones y se generarán gráficos para comparar los tiempos promedio contra el tamaño del arreglo, incluyendo las cotas superior e inferior correspondientes a cada algoritmo. Además, se graficarán los tiempos promedio en un mismo par de ejes con escala logarítmica para facilitar la comparación entre los algoritmos cuadráticos y los lineales.**

Palabras clave—ordenamiento, selección, inserción, mezcla.

I. Introducción

En este estudio se propone analizar comparativamente tres algoritmos de ordenamiento: Selección, Inserción y Mezcla. El propósito primordial radica en la evaluación y confrontación de sus tiempos de ejecución en diversos tamaños de arreglos. A través de la implementación en C++ y la realización de experimentos con arreglos de hasta 200,000 elementos, se busca brindar una comprensión detallada sobre la eficiencia relativa de cada algoritmo en distintos contextos. Esta información resulta fundamental para la selección pertinente del algoritmo acorde a las necesidades específicas de procesamiento de datos.

II. Metodología

Para la realización de esta investigación, se adoptó un enfoque metodológico estructurado que constó de varias etapas. En primer lugar, se procedió a generar arreglos aleatorios de números enteros de distintos tamaños, específicamente 50000, 100000, 150000 y 200000 elementos. Esta generación se realizó mediante una función diseñada para asignar valores aleatorios en el rango de 0 a 999 a cada elemento del arreglo.

Una vez generados los arreglos, se procedió a ejecutar los algoritmos de ordenamiento seleccionado para el estudio: selección, inserción y mezcla. Cada algoritmo se ejecutó repetidamente en los arreglos aleatorios generados, llevándose a cabo tres ejecuciones para cada tamaño de arreglo.

La medición del tiempo de ejecución de cada algoritmo se llevó a cabo utilizando la biblioteca ‘chrono’ de C++, la cual permite capturar el tiempo transcurrido entre el inicio y la finalización de la ejecución de un algoritmo. Específicamente, se registró el tiempo en milisegundos para cada ejecución de cada algoritmo en cada tamaño de arreglo.

Posteriormente, se llevó a cabo un análisis detallado de los resultados obtenidos, centrándose en la comparación de los tiempos de ejecución promedio de cada algoritmo en función del tamaño del arreglo. Con esto, buscando patrones y tendencias que permitieran determinar la eficiencia relativa de cada algoritmo en diferentes escalas de datos.

Cuadro I Tiempo de ejecución de los algoritmos

		Tiempo (ms)			
		Ejecución			
Algoritmo	Tam (n)	1	2	3	Prom.
Selección	50000	1902.95	2154.41	1447.62	1835.993
	100000	5586.58	5525.6	5560.19	5557.457
	150000	10284.2	11065.6	12537.1	11295.967
	200000	23599.6	23240	23291.8	23377.133
Inserción	50000	1517.75	1105.57	2024.59	1549.303
	100000	4020.79	3954.36	3856.17	3943.44
	150000	8738.89	8580.39	8486.97	8602.75
	200000	15365.2	15977.9	15958	15767.7
Mezcla	50000	5.18599	5.47759	5.06788	5.24382
	100000	13.3261	12.3628	10.2872	11.99203
	150000	16.3529	17.7884	17.9032	17.34816
	200000	22.5793	25.6176	23.6585	23.9518

Finalmente, los resultados de cada algoritmo se presentaron de manera clara y concisa en un archivo de texto denominado "resultados.txt". Además, se elaboraron gráficos para visualizar de forma más intuitiva la relación entre el tamaño del arreglo y el tiempo de ejecución promedio de cada algoritmo, lo que facilitó su interpretación y comprensión.

III. Resultados

El análisis de los resultados obtenidos en el Cuadro I Tiempo de ejecución de los algoritmos revela información acerca del rendimiento de los algoritmos de selección, inserción y mezcla en las diferentes configuraciones de tamaño de arreglo.

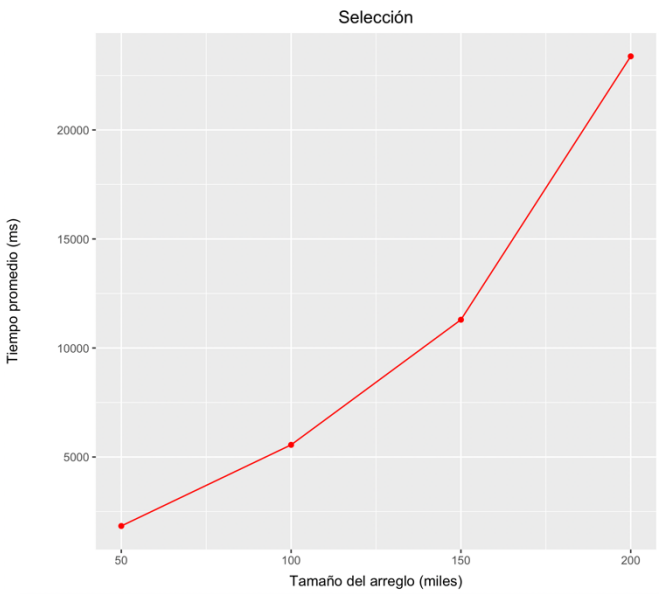
En primer lugar, al examinar el algoritmo de selección, se puede observar que los tiempos de ejecución del algoritmo muestran una variación significativa entre las tres ejecuciones para cada tamaño de arreglo. Por ejemplo, para un tamaño de 50000, los tiempos oscilan entre 1447.62 ms y 2154.41 ms, con un promedio de 1835.993 ms. Para arreglos de tamaño 200000, los tiempos varían entre 23240 ms y 23599.6 ms, con un promedio de 23377.133 ms. Esta variación sugiere que el rendimiento del algoritmo puede ser influenciado por la disposición inicial de los elementos en el arreglo, lo que impacta en el número de comparaciones y movimientos realizados. Dicho esto, en el caso promedio se confirma que tiene un orden de $\Theta(n^2)$. En términos de las cotas, superior e inferior, tanto la cota superior (O) como la cota inferior (Ω) son de orden cuadrático, lo que coincide con el comportamiento observado en los tiempos de ejecución.

Por otro lado, al analizar el algoritmo de inserción, los tiempos de ejecución de este también muestran variaciones, aunque en menor medida que el algoritmo de selección. Por

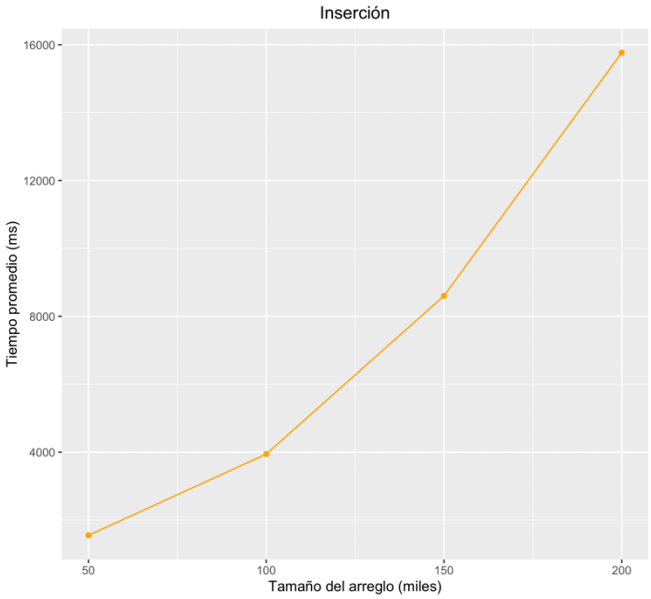
ejemplo, para un tamaño de 50000, los tiempos oscilan entre 1105.57 ms y 2024.59 ms, con un promedio de 1549.303 ms. Para arreglos de tamaño 200000, los tiempos varían entre 15365.2 ms y 15977.9 ms, con un promedio de 15767.7 ms. El crecimiento cuadrático en el algoritmo de inserción se debe a la naturaleza del algoritmo, la cual implica comparar y mover elementos a lo largo del arreglo en función de su posición y valor. A medida que el tamaño del arreglo aumenta, el número de comparaciones y movimientos necesarios también aumenta cuadráticamente, lo que resulta en un aumento exponencial en el tiempo de ejecución. Este comportamiento es consistente con la complejidad esperada del algoritmo de inserción, que realiza una cantidad cuadrática de operaciones en el peor de los casos.

En comparación con los algoritmos de selección e inserción, los tiempos de ejecución del algoritmo de mezcla muestran una variación mínima entre las diferentes ejecuciones. Por ejemplo, para un tamaño de 50000, los tiempos oscilan entre 5.06788 ms y 5.47759 ms, con un promedio de 5.24382 ms. Para arreglos de tamaño 200000, los tiempos varían entre 22.5793 ms y 25.6176 ms, con un promedio de 23.9518 ms. Tanto la cota superior (O) como la cota inferior (Ω) son de orden lineal y logarítmico (O(n log n) y Ω(n log n), respectivamente), lo que sugiere una mayor estabilidad en el rendimiento del algoritmo independientemente de la disposición inicial de los elementos en el arreglo.

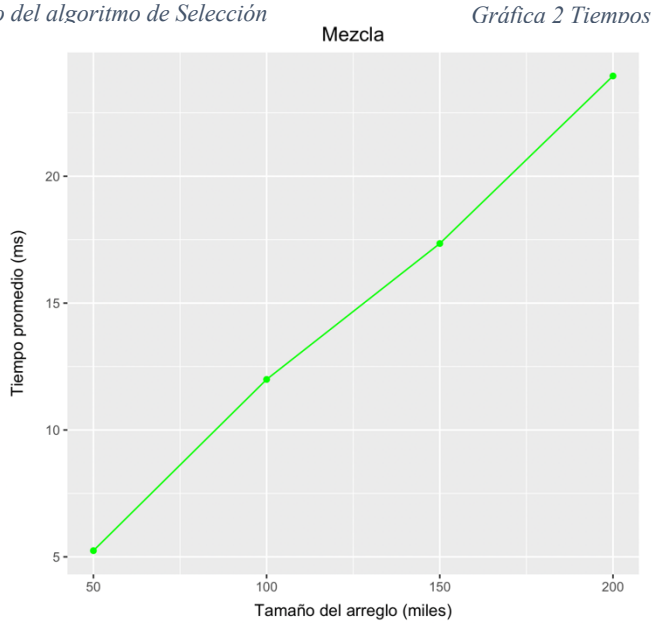
En resumen, estos análisis resaltan las diferencias en la variación de los tiempos de ejecución entre los diferentes algoritmos, así como la correspondencia entre los resultados observados y las cotas teóricas de cada algoritmo.



Gráfica 1 Tiempos promedio del algoritmo de Selección

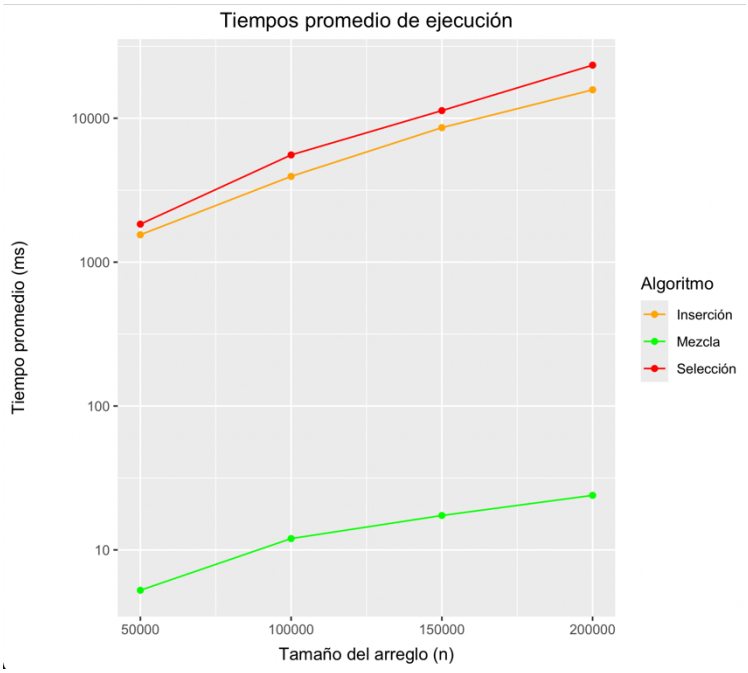


Gráfica 2 Tiempos promedio del algoritmo de Inserción



Gráfica 3 Tiempos promedio del algoritmo de Mezcla

A continuación, se muestra una gráfica con los tiempos promedio de ejecución en un mismo par de ejes.



Gráfica 4 Tiempos promedios de los algoritmos de los algoritmos de Selección, Inserción y Mezcla.

Las curvas se muestran de forma conjunta en la figura Gráfica 4. La gráfica resalta como el algoritmo de mezcla en comparación con los algoritmos de selección e inserción posee mayor control y es más estable. Mientras que estos últimos muestran un crecimiento exponencial en el tiempo de ejecución a medida que aumenta el tamaño del arreglo, el algoritmo de mezcla mantiene un crecimiento más lineal. Esta diferencia en el comportamiento refleja la capacidad del algoritmo de mezcla para mantener un control más consistente sobre el tiempo de ejecución, incluso cuando se enfrenta a arreglos de mayor tamaño. En otras palabras, a medida que los datos aumentan en complejidad y tamaño, el algoritmo de mezcla demuestra una capacidad más estable para manejar esta carga de trabajo sin experimentar un aumento drástico en el tiempo de ejecución.

IV. Conclusiones

En general, este estudio describe el rendimiento de tres algoritmos de clasificación: selección, inserción y mezcla. Al ejecutar varias pruebas con diferentes tamaños de arreglos, se pudo observar y analizar cómo cambió el tiempo de ejecución de cada algoritmo. El estudio encontró que el algoritmo de mezcla mostró una mayor estabilidad en términos de tiempo de ejecución y un crecimiento lineal más persistente en comparación con el algoritmo de selección e inserción, que mostraron un crecimiento exponencial al aumentar el tamaño del arreglo. Esto sugiere que el algoritmo de mezcla es una opción más eficiente y fiable para aplicaciones que necesitan ordenar grandes conjuntos de datos.

Estos resultados resaltan la importancia de elegir un algoritmo apropiado en función de las necesidades específicas de la aplicación, teniendo en cuenta factores como el tamaño y la complejidad de los datos a ordenar. En última instancia, esta investigación proporciona una base sólida para tomar decisiones informadas al diseñar e implementar algoritmos de clasificación en una variedad de situaciones.

Referencias

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L. y Stein, C. Introduction to Algorithms, IV ed. MIT Press, 2022.

Gael Alpízar Alfaro
Facultad de ingeniería, Computación con Varios Énfasis
Universidad de Costa Rica (UCR).
Carné C20270.

APÉNDICE A

Código de los Algoritmos

El código se muestra en los algoritmos 1, 2 y 3.

Algoritmo 1 El algoritmo de selección es un método de ordenación que busca el elemento más pequeño en cada iteración y lo coloca en la posición correcta, intercambiándolo con el elemento actual en la lista. Este proceso se repite hasta que toda la lista está ordenada. En cada iteración, busca el elemento más pequeño en la parte no ordenada y lo coloca al principio de la lista ordenada. Este proceso continúa hasta que todos los elementos están en su lugar. Aunque no es tan eficiente como otros algoritmos para grandes conjuntos de datos, el algoritmo de selección es simple y fácil de implementar.

```
    void Ordenador::seleccion(int *A, int n){
         for (int i = 0; i < n - 1; i++) {
            int m = i;
            for (int j = i + 1; j < n; j++) {
                if (A[j] < A[m]) {
                    m = j;
                }
            }
            // swap(A[i], A[m])
            int temp = A[i];
            A[i] = A[m];
            A[m] = temp;
        }
}
```

Algoritmo 2 El algoritmo de inserción es un método de ordenamiento que recorre un arreglo de izquierda a derecha, y en cada iteración, toma un elemento de la lista y lo inserta en su posición correcta en el subarreglo ya ordenado a la izquierda. El proceso de inserción se repite hasta que todos los elementos están en su posición correcta.

```
    void Ordenador::insercion(int *A, int n){
         for (int i = 1; i < n; i++) {
            int key = A[i];
            int j = i - 1;
            while (j >= 0 && A[j] > key) {
                A[j + 1] = A[j];
                j = j - 1;
            }
            A[j + 1] = key;
        }
}
```

Algoritmo 3 El algoritmo de mezcla es un método para organizar una lista de elementos. Funciona dividiendo la lista en partes más pequeñas hasta que cada parte tenga solo un elemento. Luego, combina estas partes gradualmente, asegurándose de que estén en orden mientras las une. Este proceso continúa hasta que se vuelva a formar la lista completa, pero esta vez en orden. Es como organizar un montón de cartas: primero divides las cartas en pilas pequeñas, luego las mezclas de nuevo, asegurándote de que estén en orden.

```
void Ordenador::merge(int *A, int p, int q, int r){
    int nL = q - p + 1;
    int nR = r - q;
    int *L = new int[nL];
    int *R = new int[nR];

    // Copiar elementos a los subarreglos L y R
    for (int i = 0; i < nL; i++)
        L[i] = A[p + i];
    for (int j = 0; j < nR; j++)
        R[j] = A[q + j + 1];

    int i = 0, j = 0, k = p;

    // Mezclar los subarreglos L y R en el arreglo A
    while (i < nL && j < nR) {
        if (L[i] <= R[j]) {
            A[k] = L[i];
            i++;
        } else {
            A[k] = R[j];
            j++;
        }
        k++;
    }

    // Copiar los elementos restantes de L (si hay alguno)
    while (i < nL) {
        A[k] = L[i];
        i++;
        k++;
    }

    // Copiar los elementos restantes de R (si hay alguno)
    while (j < nR) {
        A[k] = R[j];
        j++;
        k++;
    }
```

```
        // Liberar memoria
        delete[] L;
        delete[] R;
    }

    void Ordenador::mergeSortRecursive(int *A, int p, int r){
        if (p >= r)
            return;
        int q = (p + r) / 2;
        mergeSortRecursive(A, p, q);
        mergeSortRecursive(A, q + 1, r);
        merge(A, p, q, r);
    }

    void Ordenador::mergesort(int *A, int n){
        mergeSortRecursive(A, 0, n - 1);
    }

    string ImprimirDatosDeTarea(){
        return "c20270 Tarea 1 Etapa 1";
}
```