***Resumen: Este trabajo tiene como finalidad aplicar los algoritmos estudiados en clase. Para lograr esto los algoritmos fueron implementados en C++. Los algoritmos fueron probados con arreglos de números enteros aleatorios de tamaño fijo, esto con la finalidad de medir los tiempos de ejecución al ordenar dichos arreglos. Gracias a esto se logró concluir que los algoritmos de orden exponencial comparados con los algoritmos de orden nlog(n), son mucho más lentos a la hora de ordenar grandes cantidades de datos.***

1. Introducción

Los algoritmos de ordenamiento son unos de los elementos de la programación más usados en el mundo actual, estos sirven como base para el ordenamiento de grandes cantidades de datos en sistemas de bases de datos de diferentes tipos. En este trabajo se analiza la efectividad e implementación de ciertos algoritmos utilizados para este fin con la finalidad de demostrar su verdadera efectividad bajo grandes cargas de datos y la velocidad para realizar dichas tareas. Este trabajo está enfocado exclusivamente en la implementación y análisis de los algoritmos de selección, inserción y combinación.

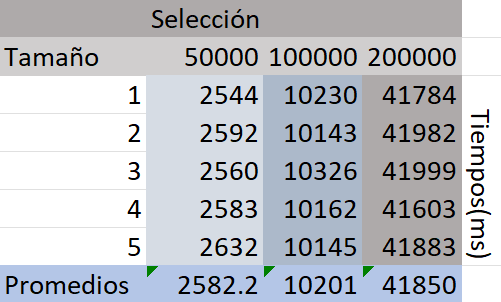
1. Metodología

Para llevar a cabo este análisis los algoritmos fueron implementados en C++, para probar su efectividad era necesario hacerlo con grandes cantidades de datos y es por este motivo que se decidió utilizar arreglos de números enteros con tamaños de 50000, 100000 y 200000 elementos. Cada uno de los algoritmos fue probado con estas 3 cantidades de datos durante 5 veces. Algo de suma importancia para sustentar la precisión de estas mediciones son las especificaciones técnicas del equipo en el cual se corrieron las pruebas.

Es así como el equipo utilizado para correr dichas pruebas es una laptop Asus X515 con un procesador i5-1135g7, 12GB de memoria RAM a 2666MHz y 256GB M.2 de almacenamiento. Además, estas pruebas fueron realizadas con la terminal de Visual Studio Code y en el momento de las mediciones se procuró minimizar los procesos en segundo plano y no había otras ventanas abiertas además de Visual Studio Code.

Para procurar mediciones de tiempo más precisas se utilizó la librería “chrono” la cual permite guardar en una variable el tiempo actual del equipo, entonces antes de realizar la llamada a cada algoritmo se tomó un tiempo inicial en una variable “S” y al terminar el llamado al algoritmo de ordenamiento se tomó nuevamente el tiempo en una variable “F”, por lo que para obtener el tiempo de ejecución del algoritmo basta con realizar la operación F-S.

Luego de aplicar las mediciones en cada uno de los algoritmos con cada uno de los distintos arreglos de enteros se obtuvo la información que se detalla en las siguientes tablas:



Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

1. Resultados

Luego de realizar las mediciones a cada uno de los algoritmos y guardar estos datos en las tablas anteriores, es importante representar estos datos en una curva grafica que nos muestre los tiempos que tomó cada algoritmo en ordenar los arreglos de distintas cantidades de elementos.

Tras analizar las gráficas podemos identificar cuáles son los algoritmos más rápidos, en este caso el algoritmo de ordenamiento por mezcla toma la delantera con una gran ventaja sobre los demás como el algoritmo más veloz para realizar estas operaciones de ordenamiento, seguido por el algoritmo de ordenamiento por inserción que es más rápido que el ordenamiento por inserción por una deferencia más pequeña.

¿A qué se deben estas diferencias? Bueno, estos algoritmos se pueden representar con modelos matemáticos que reflejan su comportamiento en orden a una función. En el caso los algoritmos de ordenamiento por selección e inserción estamos frente a algoritmos de orden Θ(n^2) por lo que al duplicar la cantidad de datos por lo general el tiempo de ejecución se va a cuadruplicar. Por otra parte, el algoritmo por mezcla gracias a su comportamiento recursivo se apega a una función de orden Θ(nlog(n)), esto hace que posea un crecimiento de la relación tiempo/tamaño mucho más controlado que en los casos anteriores.

Para poder ver como estos tiempos se comparan, se mostrará la siguiente gráfica en la cual el tiempo será el eje vertical y el tamaño de los arreglos será el eje horizontal.

Cabe destacar que para dar una mejor representación visual se utilizó una escala logarítmica.

1. Conclusiones

Gracias a los resultados obtenidos en la etapa de pruebas se llegó a la conclusión de que existen algoritmos sumamente veloces en comparación de otros, tomar esto en cuenta es algo indispensable en el desarrollo profesional de un estudiante de computación ya que en el mundo laboral esto es crucial en labores de optimización de recursos. Nos encontramos en una era en la cual la computación clásica está llegando a sus límites, por lo que poder desarrollar algoritmos cada vez más veloces es una necesidad latente para poder mantener un buen rendimiento de todos los sistemas.

Apéndice A

Código de los algoritmos en C++

**Algoritmo 1.** Ordenamiento por selección

**void seleccion**(int \*A, int n) **{**

        A = A-1;

**for**(int i = 1; i <= n; ++i) **{**

            int minIndex = i;

**for** (int j = i + 1; j <= n; ++j) **{**

**if** (A[j] < A[minIndex]) **{**

                    minIndex = j;

**}**

**}**

*//INTERCAMBIO DE ELEMENTO MENOR CON PRIMERA POSICION DEL ARRAY SIN ORDENAR*

            int temporal = A[i];

            A[i] = A[minIndex];

            A[minIndex] = temporal;

**}**

**}**

**Algoritmo 2.** Ordenamiento por inserción

**void insercion**(int \*A, int n) **{**

        A = A-1;

**for** (int i = 2; i <= n; ++i) **{**

            int key = A[i];

            int j = i-1;

**while** (j > 0 && A[j] > key) **{**

                A[j+1] = A[j];

                --j;

**}**

            A[j+1] = key;

**}**

**}**

**Algoritmo 3.** Ordenamiento por mezcla

//Parte Recursiva

**void mergesort**(int \*A, int p, int r) **{**

**if** (p >= r) **{**

            return;

**}**

        int q = ((p+r)/2);

        mergesort(A, p, q);

        mergesort(A, q+1, r);

        merge(A, p, q, r);

**}**

//Mezcla

**void 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];

**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;

        int j = 0;

        int k = p;

**while** (i < nl && j < nr) **{**

            if (L[i] <= R[j]){

                A[k] = L[i];

                ++i;

**} else {**

                A[k] = R[j];

                ++j;

**}**

            ++k;

**}**

**while** (i < nl) **{**

            A[k] = L[i];

            ++i;

            ++k;

**}**

**while** (j < nr) **{**

            A[k] = R[j];

            ++j;

            ++k;

**}**

        delete[] L;

        delete[] R;

**}**