INSTITUTO POLITÉCNICO NACIONAL

"ESCOM" (ESCUELA SUPERIOR DE CÓMPUTO)

DOCENTE: CECILIA ALBORTANTE MORATO

INTEGRANTES:

MONTEALEGRE ROSALES DAVID URIEL

VÁZQUEZ BLANCAS CÉSAR SAID

U.A.: ANÁLISIS Y DISEÑO DE ALGORITMOS

GRUPO: 3CM6

PRÁCTICA 2: Comparativa de algoritmos de ordenamiento

Introducción.

El problema de ordenamiento es uno de los problemas fundamentales en la informática, y consiste en organizar un conjunto de elementos en un orden específico. Aunque puede parecer trivial, este problema es esencial para una amplia variedad de aplicaciones, desde la clasificación de datos hasta la generación de informes y la búsqueda eficiente de información.

Existen numerosos algoritmos de ordenamiento que se utilizan en la práctica, cada uno con sus ventajas y desventajas. Algunos de los algoritmos más comunes son el ordenamiento burbuja, el ordenamiento por selección, el ordenamiento por inserción y el ordenamiento por mezcla.

La elección del algoritmo de ordenamiento adecuado depende de la aplicación específica y del tamaño del conjunto de datos que se está ordenando. Para conjuntos pequeños, los algoritmos simples como el ordenamiento por inserción pueden ser suficientes, pero para conjuntos de datos más grandes se requieren algoritmos más eficientes, como el ordenamiento por mezcla.

Objetivo de la práctica.

El propósito de la práctica de los algoritmos de métodos de ordenamiento es enseñar a los estudiantes cómo ordenar y clasificar datos de manera eficiente y efectiva mediante el uso de estos algoritmos. Dichos códigos son herramientas importantes en la programación y se utilizan para ordenar una lista de datos en un orden específico. Los datos pueden ser ordenados en orden ascendente o descendente y los algoritmos de ordenamiento pueden ser seleccionados en función de la cantidad de datos y del tipo de datos a ordenar

Desarrollo de la práctica

En esta práctica se diseñaron algoritmos para cuatro tipos de ordenamiento distintos los cuales son el ordenamiento burbuja, inserción selección y el ordenamiento por mezcla. Explicaremos cada uno de ellos y cuál es la complejidad que tiene cada uno.

1. Ordenamiento burbuja.

El método de ordenación por burbuja es el más conocido y popular entre los estudiantes y aprendices de programación debido a su facilidad para comprender y programar. Sin embargo, este método es el menos eficiente y, por lo tanto, generalmente se aprende su técnica, pero no se utiliza con frecuencia. La técnica utilizada en este método se llama ordenación por burbuja u ordenación por hundimiento, ya que los valores más pequeños "burbujean" gradualmente hacia la parte superior del array mientras que los valores mayores se hunden en la parte inferior del array. La técnica consiste en hacer varias pasadas a través del array y comparar parejas sucesivas de elementos. Si una pareja está en orden creciente, se dejan los valores como están, pero si una pareja está en orden decreciente, se intercambian los valores en el array.

Este ordenamiento consiste en ordenar una lista o array de n elementos mediante n-1 iteraciones. En cada iteración se comparan los elementos adyacentes y se intercambian si el primero es mayor que el segundo. Al final de cada iteración, el elemento mayor se sitúa en la última posición de la sub-lista actual. Una vez finalizada la primera iteración, la cola de la lista queda ordenada y el resto permanece desordenado.

```
void burbuja(int *a, int n) {
    int i, j;
    for (i = 0; i < n-1; i++) {
        for (j = 0; j < n-i-1; j++) {
            if (a[j] > a[j+1]) {
                int temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
}
```

Esto variará según la versión que se utilice. En la versión más básica, se realizan n-1 pasadas y n-1 comparaciones en cada una. Por lo tanto, el número total de comparaciones es $(n-1)*(n-1) = n2-2n+1$, lo que significa que la complejidad es de orden O (n^2).

2. Ordenamiento por inserción.

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

El procedimiento de clasificación mediante el método de inserción es semejante a la técnica comúnmente utilizada para ordenar tarjetas de nombres o cartas de una baraja, en la cual se introduce el nombre correspondiente en el lugar adecuado dentro de una lista o archivo que ya ha sido organizado previamente por orden alfabético.

La función “insercion” está compuesta por dos argumentos: el arreglo A y la dimensión de la lista n. El algoritmo respectivo consta de los siguientes pasos: En primer lugar, se considera que el primer elemento de A, A [0], ya se encuentra ordenado, por lo que la lista inicial tiene solamente un elemento. En segundo lugar, se inserta el elemento A [1] en la ubicación apropiada, ya sea antes o después de A [0], dependiendo de si es mayor o menor. Para lograr esto, se explora la lista desde A[i] hasta A[n], buscando la posición correcta para insertar el elemento dentro de la lista ya ordenada. En tercer lugar, en cada iteración del bucle, se mueve cada elemento mayor a la posición a insertar hacia abajo (a la derecha en la lista) una posición, a fin de dejar disponible dicha posición. Por último, se inserta el elemento en la posición adecuada.

La técnica de ordenación por inserción tiene una cantidad establecida de pasos. En una pasada determinada, la inserción tiene lugar en una sub-lista que va desde A [0] hasta A[i], y se necesitan en promedio i/2 comparaciones para completar dicha inserción. A diferencia de otras técnicas, el procedimiento de ordenación por inserción no emplea intercambios. La complejidad del algoritmo, que mide la cantidad de comparaciones necesarias, es de O (n^2).

3. Ordenamiento por selección.

Este es un algoritmo para ordenar un array de enteros A en orden ascendente, es decir, del número más pequeño al más grande. Si el array A tiene n elementos, el objetivo es reordenar los valores de la lista de manera que A [0] contenga el valor más pequeño, A [1] el siguiente valor más pequeño, y así sucesivamente hasta que A[n-1] contenga el valor más grande. El algoritmo se basa en repetidas pasadas que intercambian sucesivamente el elemento más pequeño con el primer elemento de la lista A [0]. De esta manera, se busca el elemento más pequeño de la lista y se intercambia con A [0], el primer elemento de la lista.

[A] A [0] A [1] A [2].... A[n-1]

Luego de la primera pasada en el algoritmo de ordenamiento, el primer elemento del arreglo está ordenado y el resto de la lista, A [1], A [2]... A[n-1], permanece desordenado. En la siguiente pasada, se busca el elemento más pequeño en esta lista desordenada y se lo coloca en la posición A [1]. De esta manera, los elementos A [0] y A [1] están ordenados, y la sub-lista A [2], A [3] ... A[n-1] permanece desordenada.

```
void seleccion(int *a, int n) {
    int i, j, aux,k;
    for (i = 0; i < n-1; i++) {
        aux=a[i];
        k=i;
        for (j = i+1; j < n; j++) {
            if (a[j] < aux) {
                aux=a[j];
                k=j;
            }
        }
        a[k]=a[i];
        a[i]=aux;
    }
}
```

El proceso se repite n-1 veces, en cada pasada se selecciona el siguiente elemento más pequeño de la sub-lista desordenada y se intercambia con el primer elemento de la sub-lista. En la última pasada, la lista desordenada se reduce a un elemento (el mayor de la lista) y el arreglo completo queda ordenado.

La complejidad del algoritmo se mide por el número de comparaciones y es cuadrática, es decir O (n^2); el número de intercambios es $O(n)$. No hay caso mejor ni peor dado que el algoritmo realiza un número fijo de pasadas y explora o rastrea un número especificado de elementos en cada pasada.

4. Ordenamiento por mezcla.

El procedimiento descansa en la noción siguiente: Si la lista tiene una longitud pequeña, siendo esta vacía o conteniendo solo un elemento, ya se considera ordenada, por lo que no se requiere realizar ninguna acción. En caso contrario, se debe ejecutar lo siguiente:

a. Separar la lista en dos mitades, obteniendo dos sub listas de tamaños semejantes (aproximadamente).
b. Ordenar cada una de las sub listas por medio de este mismo procedimiento.
c. Una vez que ambas sub listas se encuentren ordenadas, se deben intercalar de forma ordenada.

El Ordenamiento por Mezcla es considerado uno de los algoritmos de ordenamiento más eficaces y ampliamente utilizados. Este método se basa en la técnica de dividir y conquistar, dividiendo la lista de elementos en dos partes iguales de manera repetida hasta que se llega a una lista de elementos individuales, los cuales ya se consideran ordenados por sí mismos. Luego, el algoritmo combina estas pequeñas listas ordenadas una y otra vez, para generar listas cada vez más grandes y ordenadas, hasta llegar a obtener la lista final ordenada. Este método es muy popular en el ámbito del comercio electrónico.

```c
void merge(int *a, int p, int q, int r) {
    int n1 = q - p + 1;
    int n2 = r - q;
    int *L = (int *)malloc(n1 * sizeof(int));
    int *R = (int *)malloc(n2 * sizeof(int));

    for (int i = 0; i < n1; i++) {
        L[i] = a[p + i];
    }
    for (int j = 0; j < n2; j++) {
        R[j] = a[q + j + 1];
    }

    int i = 0, j = 0, k = p;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            a[k] = L[i];
            i++;
        } else {
            a[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        a[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        a[k] = R[j];
        j++;
        k++;
    }

    free(L);
    free(R);
}
```

Para ordenar un vector, el algoritmo de ordenamiento por combinación lo divide en dos sub vectores de igual tamaño, ordena cada sub vector y después los combina en un vector más grande. Con un número impar de elementos, el algoritmo crea los dos sub vectores de tal forma que uno tenga más elementos que el otro. La implementación es recursiva.

```c
void mergeSort(int *a, int p, int r) {
    if (p < r) {
        int q = (p + r) / 2;
        mergeSort(a, p, q);
        mergeSort(a, q + 1, r);
        merge(a, p, q, r);
    }
}
```

El paso básico es un vector con un elemento que, desde luego, está ordenado, por lo que el ordenamiento por combinación regresa de inmediato cuando se le llama con un vector de un elemento. El paso recursivo divide a un vector de dos o más elementos en dos sub vectores de igual tamaño, ordena en forma recursiva cada sub vector y después los combina en un vector ordenado de mayor tamaño.

La complejidad del algoritmo en su caso promedio es de O (nlog(n)).

Resultados.

1. Ordenamiento burbuja.

```
Escribe los valores con los que quieres trabajar: 1000000

Tiempo de ejecucion de burbuja: 5866.104000 segundos
```

El algoritmo burbuja soporto una carga de un millón de datos y fue en un tiempo de 1 hora y 30 min aproximadamente.

2. Ordenamiento inserción.

```
Escribe los valores con los que quieres trabajar: 1000000

Tiempo de ejecucion de insercion: 1735.007000 segundos
```

El algoritmo inserción soporto una carga de un millón de datos y fue en un tiempo de 30 min aproximadamente.

3. Ordenamiento selección.

```
Escribe los valores con los que quieres trabajar: 1000000

Tiempo de ejecucion de seleccion: 2171.549000 segundos
```

El algoritmo selección soporto una carga de un millón de datos y fue en un tiempo de 40 min aproximadamente.

4. Ordenamiento mezcla.

```
Escribe los numeros con los que quieres trabajar: 1000000000
Tiempo de ejecucion de merge: 1036.497000 segundos
```

El algoritmo merge soporto una carga de 1000000000 de datos y fue en un tiempo de 30 min aproximadamente.

TABLA DE RESULTADOS

Número de datos en la entrada	Burbuja	Selección	Inserción	Mezcla
10	0.0 s	0.0 s	0.0 s	0.0 s
100	0.0 s	0.0 s	0.0 s	0.001 s
1,000	0.007 s	0.001 s	0.0 s	0.001 s
10,000	0.522 s	0.192 s	0.159 s	0.006 s
40,000	8.358 s	3.212 s	2.559 s	0.025 s
100,000	51.612 s	18.903 s	15.44 s	0.077 s
200,000	207.013 s	75.26 s	60.77 s	0.149 s
400,000	832.917 s	324.412 s	251.45 s	0.269 s
600,000	2141.08 s	755.982 s	624.194 s	0.399 s
1,000,000	5866.104 s	2171.549 s	1735.007 s	0.676 s