Tarea 2: Robustez de Ordenamiento

Primavera 2022

Profesor Fernando Esponda Darlington

Instituto Tecnológico Autónomo de México

Estructuras de Datos Avanzadas

Javier Nieto Merodio

1. Introducción

En esta tarea se pretende dar una propuesta para medir el error de los algoritmos de ordenamiento. Para esto, se enseña una forma de simular un error en el ordenamiento de un algoritmo de ordenamiento que sí funciona. Una vez hecho esto, se medirá el nivel de ordenamiento de cinco distintos algoritmos: Insertion Sort, Selection Sort, Quick Sort, Merge Sort y Bubble sort.

2. Medición del error

Para hacer la medición del error bajo una simulación del error, se usó un arreglo de tipo objeto Películas. La clase Películas implementa la interfaz Comparable. Entonces, la facilidad de realizar el experimento es mucho mayor, ya que en vez de alterar el compareTo de cada algoritmo de ordenamiento, solo se cambia en la clase a ordenar.

2.1. Cómo simular el error

La forma de medir el error fue la siguiente. En la clase Películas, en el compareTo, se hizo una alteración. Primero, se generó un número aleatorio tipo float entre el 0 y el 1. Ahora bien, para simular el error, decimos que si el número aleatorio es mayor a la probablidad que queramos asignarle al error, que el compareTo sea ejecutado de forma correcta. En caso contrario, que el compareTo sea ejecutado de forma inversa.

En la Figura 1, enseñamos el método compareTo de películas (en este caso, la probabilidad asignada al error fue de 10%).

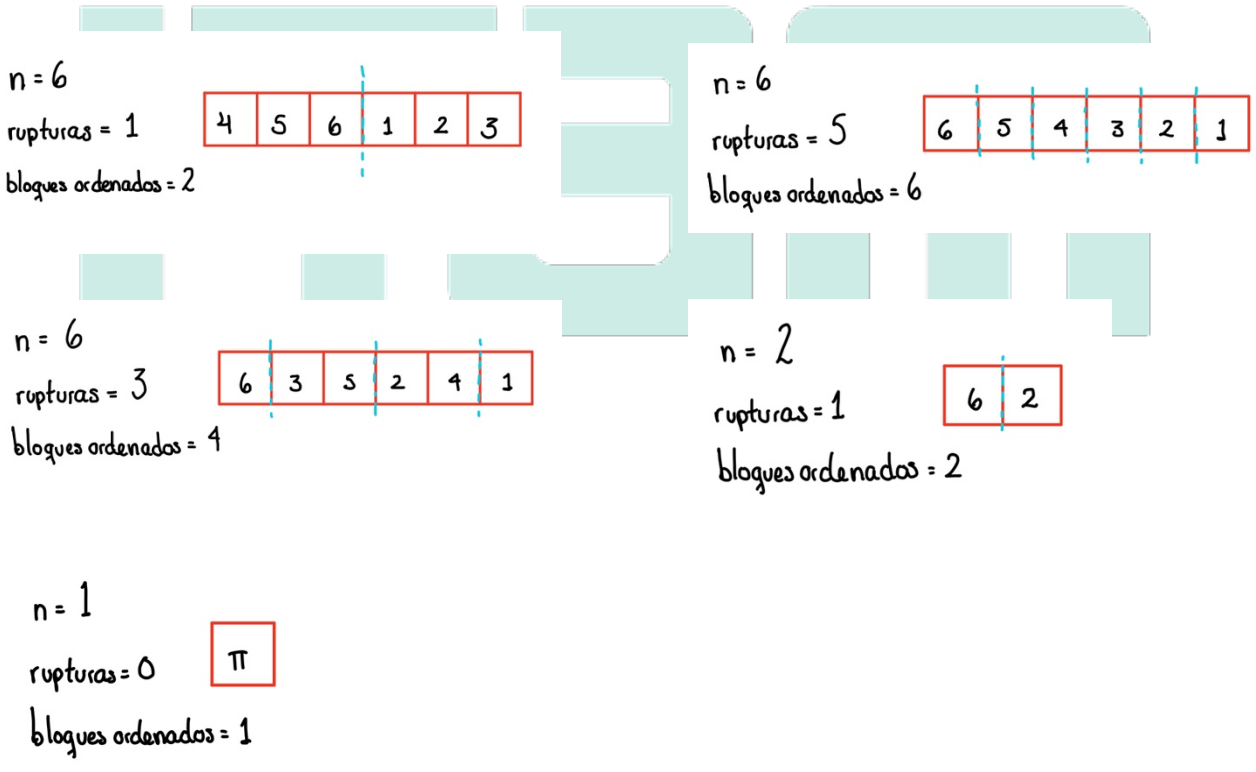
```
public int compareTo(Peliculas o) {
    Random rand = new Random();

    float prob = rand.nextFloat();

    if (prob > 0.1){
        if (this.clave > o.clave)
            return 1;
        else if (this.clave == o.clave)
            return 0;
        else
            return -1;

    }

    else{
        if (this.clave > o.clave)
            return -1;
        else if (this.clave == o.clave)
            return 0;
        else
            return 1;

    }


}
```

Figura 1: compareTo para simulación de ruido

2.2. Propuesta de medición del error/orden de un arreglo

Ahora bien, ofreceremos una propuesta de medir un arreglo. El objetivo de un arreglo es hacer la búsqueda lo más eficiente posible. Pero ¿y si el arreglo no está ordenado totalmente? Entonces, un objetivo muy viable, sería buscar bloques de arreglo que estén ordenados. Entre menos bloques ordenados de arreglos haya, mejor. Por ejemplo, un arreglo que esté totalmente ordenado tiene un bloque singular ordenado. Nuestro objetivo sería que el número de bloques ordenados sea lo más cercano a 1.

Entonces, nuestro medidor de orden calcularía la cantidad de veces que arreglo[i].compareTo(arreglo[i + 1]) > 0. Es decir, contar cuántas veces el predecesor de un elemento en el arreglo es mayor al siguiente. Luego, al saber cuántas veces sucede esto, sumamos a la cantidad de rupturas de orden un 1 para terminar teniendo la cantidad de bloques ordenados que existen en un arreglo. Para facilitar la explicación, se expondrán distintos ejemplos:



Ahora bien, esta forma de ordenamiento tiene sus ventajas y sus desventajas. Como ventaja, a ciertas aplicaciones le facilitaría el trabajo al saber qué tan fácil es buscar, ya que entre más bloques ordenados haya, más difícil será la búsqueda. Por otro lado, un caso problemático sería el arreglo que vaya totalmente en orden inverso, que se vería de la siguiente forma: número de bloques

ordenados = longitud del arreglo. Pero es cierto que, si el número de bloques ordenados tiende a ser el tamaño de entrada, puede que el arreglo esté ordenado de forma inversa.

3. Experimento y resultados

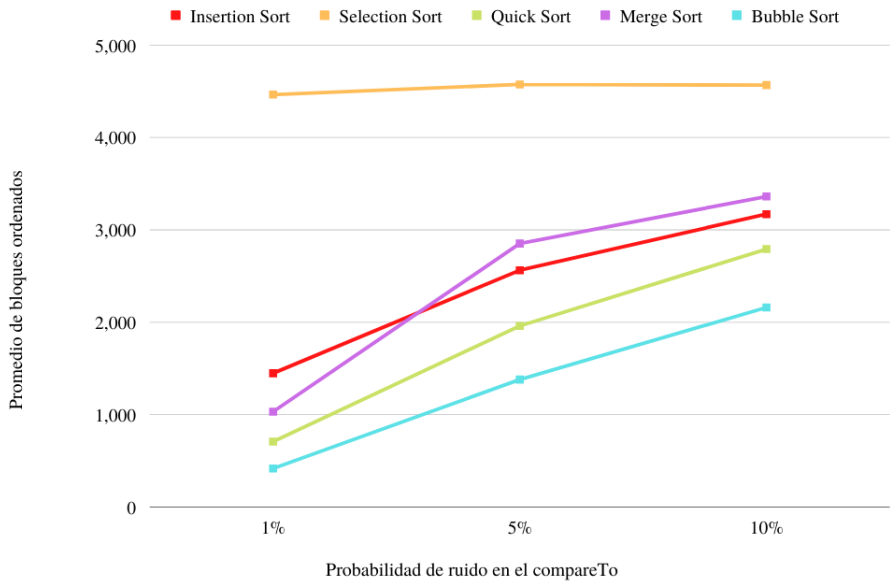
3.1. Forma de realizar el experimento

Para el experimento, se calculó qué tan bien se comportan con una probabilidad dada de error cinco distintos algoritmos de ordenamiento: Insertion Sort, Selection Sort, Quick Sort, Merge Sort y Bubble Sort. Para tener resultados más precisos, se promediaron por cada algoritmo 30 repeticiones del ordenamiento con probabilidad de error de un arreglo desordenado. Además, se calculó todo esto con tres probabilidades de error distintas: 1%, 5% y 10%. El número de entrada fue de 10,000 películas para todos los casos.

3.2. Los resultados

A continuación, mostraremos los resultados en una sola gráfica. En el eje x, se delimitarán las tres distintas probabilidades medidas. En el eje y, el número de bloques ordenados. El objetivo es que se acerquen lo más posible al 1 (para efectos visuales, al eje x). La distancia vertical entre cada línea que marque un algoritmo de ordenamiento distinto nos muestra la diferencia entre ellas de manejo ante el ruido.

Figura 2: Promedio de bloques ordenados con probabilidad de error en un arreglo de 10,000 entradas

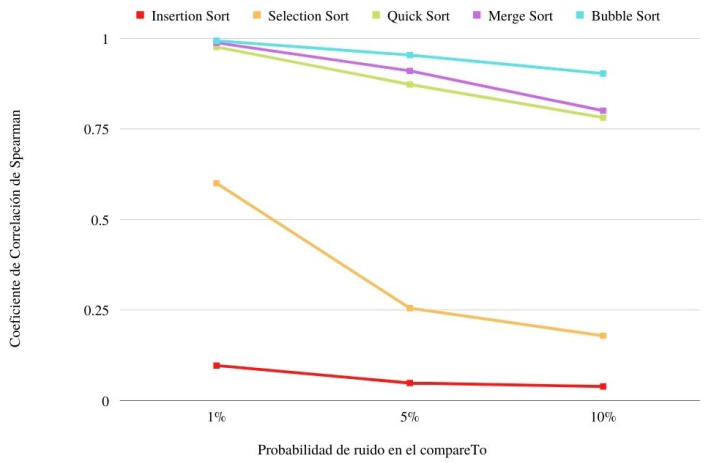


La gráfica nos muestra que, ante el medidor de orden propuesto, el Bubble Sort es el que mejor se comporta. Lo más probable es que esto se deba a un número extenso de compareTo´s por ejecución del método. Así, de haber un error, hay mayor probabilidad de haber ordenado otros bloques. Los demás algoritmos, se comportan de manera similar, a excepción del Merge Sort. Este algoritmo de ordenamiento parece tener una tendencia a mantener el número de bloques ordenados cercanos al 50% del tamaño de entrada. Este es el peor caso, ya que no tiende a un bloque ordenado ni a uno ordenado de forma inversa.

3.3. Comparación de los resultados con el Coeficiente de correlación de Spearman

Por último, compararemos los resultados antes presentados con una forma de medir el ordenamiento bastante popular: el coeficiente de correlación de Spearman. Este mide el coeficiente de correlación entre dos arreglos distintos. El objetivo es que el coeficiente de correlación entre el arreglo perfectamente ordenado y el ordenado con probabilidad de ruido tienda a 1. Entre más cerca esté de 1, más cerca está de tolerar la probabilidad de error. Los resultados de aplicar Spearman a un arreglo Películas (tamaño de entrada de 10,000) ordenado con los cinco algoritmos de ordenamiento con probabilidad de error de 1%, 5% y 10% son los siguientes:

Figura 3: Coeficiente de Correlación de Spearman en métodos de ordenamiento con probabilidad de error (10,000 entradas)



Los resultados nos muestran algo claro: Bubble Sort, al ser el más cercano al coeficiente de 1, es el mejor ordenado. Independientemente del método usado para medir la tolerancia al ruido, este algoritmo de ordenamiento sigue siendo el más eficiente. Por otro lado, el orden de peor a mejor

algoritmo si cambia. Esto nos demuestra que, si bien, Bubble Sort parece ser el más eficiente, los métodos de medición de robustez sí muestran distintos resultados entre sí.

4. Conclusión

En conclusión, ante la presencia de ruido en un compareTo, los métodos de ordenamiento que mejor se comportan puede que sean los que más se tarden. Esto, ya que presentan un número mayor de comparaciones, y toleran mejor el tener un elemento no ordenado en el arreglo. Con base en los resultados, Bubble Sort es el que mejor tolera la presencia de ruido en las comparaciones. Por la smilitud de los resultados entre el método de ordenamiento por bloques y el de Spearman, Bubble Sort parece predominar en tolerancia al ruido independientemente de qué propuesta de ordenamiento se use.