Tarea 6: Comparación de algoritmos de ordenación

Profesores: Nelson Baloian
Patricio Poblete
Auxiliares: Manuel Cáceres
Sebastián Ferrada
Sergio Peñafiel

Fecha de Entrega: Viernes 24 de Junio 23:59hrs

1 Introducción

El objetivo de esta tarea es comparar empíricamente los tiempos de ejecución de diferentes algoritmos de ordenación basados en comparación vistos en el curso.

En particular pretendemos investigar la diferencia entre:

- Diferentes algoritmos $\mathcal{O}(n \log n)$.
- Diferentes variantes del algoritmo QuickSort.

Para esto utilizaremos los algoritmos MergeSort y 3 variantes de QuickSort, aplicados sobre arreglos de enteros de n elementos que contienen una permutación al azar de $\{1, 2, \ldots, n\}$.

MergeSort

Este algoritmo separa el conjunto a ordenar en dos subconjuntos de tamaño similar, ordena estos subconjuntos por separado para luego ordenar el conjunto total en un proceso denominado "merge" de estos subconjuntos.
Más concretamente este algoritmo divide el arreglo en 2 mitades, ordena recursivamente cada una de éstas mitades y las combina haciendo un "merge" de estas.
El número de comparaciones del algoritmo es $\mathcal{O}(n \log n)$.

QuickSort

Este algoritmo elige un elemento del conjunto a ordenar denominado "pivote", generando un subconjunto de elementos menores al pivote y otro de elementos mayores al pivote en un proceso denominado "particionar", para finalmente ordenar cada uno de estos subconjuntos.
Más concretamente estudiaremos 3 variantes incrementales de este algoritmo que tienen un número de comparaciones de $\mathcal{O}(n \log n)$ en el caso promedio.

Método clásico

Para ordenar el arreglo se elige un "pivote" al azar poniendo los elementos menores al pivote a la izquierda del arreglo y los mayores a la derecha, ordenando recursivamente cada uno de los subarreglos generados.

Mediana de tres

Análogo al método clásico, con la excepción que el "pivote" se escoge como la mediana de tres elementos escogidos al azar.

Mediana de tres e InsertionSort

Análogo al método mediana de tres, excepto que la recursión del algoritmo se detiene cuando se quiere ordenar un arreglo de tamaño menor a M (en esta tarea usaremos $M = 10$). Finalmente como este algoritmo no deja ordenado el arreglo, al final se aplica el algoritmo InsertionSort.
Es importante que el algoritmo utilizado en la fase final corresponda a InsetionSort y no a otro algoritmo $\mathcal{O}(n^2)$.

2 Implementación

Su trabajo consiste en implementar los cuatro algoritmos de ordenación : MergeSort, QuickSort (método clásico), QuickSort (mediana de tres), QuickSort (Mediana de tres e InsertionSort) previamente explicados.

Para realizar esto se espera que entregue cuatro clases que implementen la siguiente interfaz, con cada uno de los algoritmos anteriores :

```
1 public interface Ordenacion {
2   public void ordenar(int[] a);
3 }
```

3 Experimentación

Una vez implementados los algoritmos de ordenación, se le pide que compare los tiempos de ejecución de estos, para lo cual tenga en cuenta lo siguiente :

- Ejecute las pruebas en un solo computador y especifique en su informe las características de éste que usted considere pertinentes.
- Solo mida el tiempo del algoritmo y no el de otras operaciones (como la creación de las permutaciones), sea cuidadoso con esto pues sus mediciones se podrían ver alteradas.
- Se recomienda considerar arreglos de tamaño $n \in \{10^7, \ldots, 10^8\}$.
- Para generar las permutaciones al azar utilice la siguiente función que dado un n retorna un arreglo que contiene una permutación al azar de $\{1, 2, \ldots, n\}$:

```java
import java.util.Random;
...
static public int[] permutacion(int n) {
    int[] a = new int[n];
    for (int i = 0; i < n; ++i)
      a[i] = i+1;

    Random rnd = new Random();
      for (int i = 0; i < n; i++) {
          int r = i + rnd.nextInt(n-i);
          int temp = a[i];
          a[i] = a[r];
          a[r] = temp;
      }
      return a;
}
```

Con los datos obtenidos realice gráficos que le ayuden a comparar la *performance* de los diferentes algoritmos, en particular preocúpese de que sus gráficos :

- Muestren el comportamiento asintótico correspondiente de cada algoritmo.
- Permitan diferenciar :
  - MergeSort de QuickSort.
  - Las distintas versiones de QuickSort.

4 Condiciones de Entrega

- Esta tarea debe ser resuelta en Java.
- Es obligatorio la entrega de un informe en formato pdf junto con su tarea (Ver siguiente sección).
- Esta tarea es de carácter individual, cualquier caso de copia se evaluará con la nota mínima.
- No olvide subir a U-cursos todos los archivos necesarios para que su tarea funcione correctamente.
- Debe subir los archivos de código fuente (*.java). Los archivos compilados (*.class) no serán evaluados.
- Cualquier duda respecto a la tarea puede ser consultada usando el foro del curso.
- **NO** se aceptarán atrasos.

5 Informe

El informe debe describir el trabajo realizado, la solución implementada, los resultados obtenidos y las conclusiones o interpretaciones de estos. Principalmente debe ser breve, describiendo cada uno de los puntos que a continuación se indican:

- **Portada:** Indicando número de la tarea, fecha, autor, email, código del curso, etc.
- **Introducción:** Descripción breve del problema y su solución.
- **Análisis del Problema:** Exponga en detalle el problema, los supuestos que pretende ocupar, casos de borde y brevemente la metodología usada para resolverlo. Indique también sus hipótesis preliminares.
- **Solución del Problema:** Indique claramente los pasos que siguió para llegar a la solución del problema. Muestre mediante figuras y ejemplos qué es lo que realiza su código. Evite copiar todo el código fuente en el informe, sin embargo, puede mostrar las partes relevantes de éste.
- **Resultados y Conclusiones:** Muestre los gráficos realizados y utilícelos para ilustrar sus conclusiones. Sea detallado en explicar qué partes indican tendencias en complejidad. Sea riguroso en las visualizaciones: verifique que los datos representados se ajusten a los datos obtenidos y no a lo que ud. espera mostrar.