CC3001 Tarea 5 Primavera 2018
`Huffman Sort`

Profs. Jérémy Barbay / Patricio Poblete

Fecha de entrega: Lunes 17 de Diciembre de 2018 a las 23:30 horas

El objetivo de esta tarea es implementar diferentes variantes del algoritmo de ordenamiento `Merge Sort`. En particular estudiaremos `Adaptive Merge Sort`, y `Huffman Sort`, y comparar sus rendimientos.

Para definir propiamente tales algoritmos, comenzaremos por definir el concepto de *run* sobre un arreglo $\mathcal{A}$, para lo que necesitaremos algunos conceptos previos.

Un subarreglo de $\mathcal{A}$ es una secuencia de elementos contiguos en $\mathcal{A}$. Por ejemplo, $[2, 1]$ es un subarreglo del arreglo $\mathcal{A} = [3, 2, 1, 4, 5]$, mientras que $[2, 4]$ no lo es. Note que $\mathcal{A}$ es un subarreglo de $\mathcal{A}$.

Un subarreglo es creciente si está ordenado de forma creciente, como $[1, 4]$, a diferencia de $[2, 1, 4]$ que no es creciente.

Diremos que un subarreglo creciente es maximal si no puede ser extendido y seguir siendo creciente. Por ejemplo, $[1, 4]$ es un subarreglo creciente, pero no es maximal ya que puede ser extendido a $[1, 4, 5]$, que es también creciente. $[1, 4, 5]$ si es un subarreglo creciente maximal.

Un *run* de $\mathcal{A}$ se definirá como un subarreglo creciente maximal de $\mathcal{A}$. Siguiendo con el ejemplo anterior, los runs serían $[3]$, $[2]$ y $[1, 4, 5]$.

Ahora podemos definir de manera sencilla los distintos algoritmos de ordenamiento a estudiar.

- `Merge Sort` es el algoritmo clásico de ordenamiento ("*bottom-up*", o decir, juntando los elementos dos a dos[1]);

[1] La versión "*Top-Down*", cual consiste a dividir el arreglo en dos recursivamente y después juntar las mitades, hace un uso mas grande de la pila del sistema operativo, por lo cual no se usa mucho.

- `Adaptive Merge Sort` es una variante de `Merge Sort` donde 1) se detectan los *runs* y 2) se unen (*merge*) los *runs*, tomando en casa paso los dos *runs* creados hace más tiempo. Note que los *runs* iniciales del arreglo se crean de izquierda a derecha. El resultado de esa unión será un nuevo *run* a considerar.

- `Huffman Sort` es una variante de `Merge Sort` donde 1) se detectan los *runs* y 2) se unen (*merge*) los *runs* tomando en cada paso los dos *runs* de menor tamaño (en caso de empates, se privilegiará el que comience más a la izquierda). El resultado de esa unión será un nuevo *run* a considerar.

La Tarea

Usted debe escribir un programa en Java `evaluaHuffmanSort.java`, que tome de la entrada estándar líneas constituidas de números enteros separados por espacios, y que imprime en la salida estándar las estadísticas siguientes:

- cantidad de *runs* en el input;

- cantidad de comparaciones efectuadas por `Merge Sort` entre elementos de input;

- cantidad de comparaciones efectuadas por `Adaptive Merge Sort` entre elementos de input;

- cantidad de comparaciones efectuadas por `Huffman Sort` entre elementos de input.

No se cuentan las comparaciones para elegir los *runs* más cortos, pero si se cuentan las comparaciones efectuadas para encontrar los *runs*, tanto para Adaptive Merge Sort como para Huffman Sort.

Note que por cada línea de entrada debe imprimir exactamente una línea de salida. Por ejemplo:

input	output
10 3 5 9	2 5 6 6
4 9 5 6	2 5 6 6
10 6 6 6	2 5 6 6
1 6 7 9	1 4 3 3
5 8 9 9	1 4 3 3

Afeed

El nombre de esta tarea en afeed es `huffmanSort`. Por lo tanto, para probar su código, si este se llama `evaluaHuffmanSort.java`, debe utilizar la línea `afeed huffmanSort evaluaHuffmanSort.java` en la consola, desde la carpeta conteniendo su código.

Los casos fueron diseñados cuidadosamente para no tener operaciones ilegales, pero si está convencido de que su tarea es correcta y la herramienta dice lo contrario, por favor escriba a `bernardosubercaseaux@gmail.com`.

Formato del informe

La entrega de la tarea debe incluir un informe donde se describa y documente el programa realizado. El informe (en formato PDF) y el código fuente deben ser entregados a través del sistema U-Cursos hasta las 23:30 de la fecha de entrega.

El informe debe tener una extensión máxima de cinco páginas, sin incluir la portada y anexos. El contenido del informe debe seguir la siguiente pauta referencial, común para todas las tareas restantes del semestre (vea qué partes son relevantes en cada tarea):

- Portada.
- Introducción: Describir brevemente el problema y su solución.
- Diseño de la solución: Indicar la metodología utilizada para resolver el problema, casos de borde y supuestos utilizados. Detallar la solución propuesta, describiendo el algoritmo y todas sus partes relevantes (por ejemplo invariantes, uso de recursión, etc.). Incluya figuras si es necesario.
- Implementación: Se debe mostrar la parte relevante del código del programa que soluciona el problema (omita los detalles que no tengan relevancia), explicando lo que hace el código. Se recomienda usar nombres representativos en las variables. NOTA: El código generado para resolver la tarea debe corresponder al diseño descrito, preocúpese de realizar los comentarios que sean necesarios.
- Resultados y conclusiones: Incluya resultados experimentales solicitados y conclusiones obtenidas.
- Anexo: Incluya un listado con el programa completo que se compiló realmente (utilice una fuente pequeña (8 pt) en este listado). De ser necesario, cualquier información adicional se debe agregar en los anexos y debe ser referenciada en alguna sección del informe de la tarea.

Reglas de colaboración

Discutir las tareas con sus compañeros esta autorizado, pero no compartir código. Los alumnos pueden usar código disponible públicamente solamente si citan la fuente de manera explicita, en su código como en su informe. Los alumnos no pueden hacer el código de su tarea publico, y no pueden usar la tarea de un compañero como fuente, aun que sea documentado.