Práctica 10
Ordenamientos: Parte I

1 Meta

Que el alumno conozca algunos algoritmos de ordenamiento y los implemente para ordenar arreglos de enteros.

2 Objetivos

Al finalizar la práctica el alumno:

- Será capaz de implementar en Java los algoritmos descritos en alto nivel.
- Tendrá las bases para implementar otros algoritmos de ordenamiento para arreglos de enteros.
- Comprenderá la importancia de la complejidad en tiempo de los algoritmos.

3 Antecedentes

Un algoritmo de ordenamiento tiene como objetivo ordenar las entradas de una estructura de datos. En este caso vamos a ordenar los valores de un arreglo de enteros. Para lograr esto vamos a ver tres algoritmos, dos de ellos en esta primera parte.

3.1 Bucket Sort

Bucket Sort es un algoritmo que se basa en **contar** las veces que aparece cada elemento del arreglo. La cuenta se va a guardar en un arreglo **auxiliar**. Para poder hacer esto, el algoritmo necesita como entrada, además del arreglo a ordenar (A), dos números: el valor mínimo (m) y el valor máximo (M) que pueden tomar los valores del arreglo.
Así, para poder guardar la cuenta de todos las posibles valores de las entradas de A, se necesita un arreglo auxiliar de tamaño $M - m + 1$, donde la primer entrada ($aux[0]$) corresponde al número de veces que aparece m en A; la segunda entrada $aux[1]$ al número de veces que aparece $m + 1$ y así

sucesivamente. Por ejemplo, si $m = -2$ y $M = 4$, nuestro arreglo auxiliar sería:

Índice	0	1	2	3	4	5	6
Representa la cuenta del valor:	-2	-1	0	1	2	3	4

Este arreglo auxiliar debe iniciar con todas sus entradas en 0, y sus valores se actualizarán mientras se recorre el arreglo A. Cada vez que se vea en el arreglo A el valor i tenemos que sumar 1 a la entrada del arreglo auxiliar que representa la cuenta de i. Este índice sería $i - m$.
Por ejemplo, suponga que estamos recorriendo el arreglo A:

3	4	3	-1	0	**-1**	1	0	2	3	-1

Y actualmente estamos en el índice 5, cuya entrada está resaltada. En este punto de la ejecución, el algoritmo sólo ha visto aquellos valores de A que están **antes**. Es decir ha visto dos veces 3, y una vez a $4, -1$ y 0. Entonces el arreglo auxiliar antes de actualizar la información con el -1 que estamos viendo debería ser:

Índice:	0	1	2	3	4	5	6
Representa:	-2	-1	0	1	2	3	4
Valor	0	1	1	0	0	2	1

Y después de actualizar el arreglo con la información de la posición actual:

Índice:	0	1	2	3	4	5	6
Representa:	-2	-1	0	1	2	3	4
Valor	0	2	1	0	0	2	1

Es importante que note que la información de la fila **Representa:** en las tablas anteriores no existe en el arreglo; éste únicamente contiene la información de la última fila.
Continuando con el ejemplo, cuando hayamos terminado de recorrer la entrada A, el arreglo auxiliar quedaría como:

Representa:	-2	-1	0	1	2	3	4
Valor	0	3	2	1	1	3	1

Con esta información ya podemos reescribir el arreglo A de una forma en la que esté ordenado. Simplemente vamos a escribir en el arreglo A cada

uno de los valores posibles, el número de veces que los contamos en A. Es decir, escribimos -2 cero veces; -1 tres veces; 0 dos veces y continuamos de esta forma hasta escribir todos los posibles valores. El arreglo resultante es:

-1	-1	-1	0	0	1	2	3	3	3	4

Resumiendo, **Bucket Sort** con entradas A, m y M:

1. Crea un arreglo auxiliar aux de tamaño $M - m + 1$.
2. Recorre el arreglo A, y por cada entrada con valor x, aumenta en 1 el valor de $aux[x - m]$. (Como el índice 0 de aux corresponde a m, tenemos que restar m a x para saber qué entrada le corresponde en el arreglo).
3. Crea un índice i que nos dice cuántos números hemos escrito en A, inicializado con un valor de 0.
4. Para cada índice j de aux:
   - Si $aux[j] = 0$ pasa a la siguiente entrada de aux.
   - En caso contrario, $aux[j] = n$, escribimos en n índices de de A, iniciando desde i, el valor $j + m$, (que es el valor cuya cuenta está guardada en el índice m). Por cada escritura que hagamos debemos sumar 1 al índice i.

3.2 Selection Sort

La idea de Selection Sort es en cada paso encontrar el mínimo de los elementos que no están ordenados. Esto lo hace de la siguiente forma:
Inicia en la posición 0 de tu arreglo. En este punto se asume que **todo** el arreglo está desordenado. Se busca el mínimo en todo el arreglo.

4	10	8	1	5	2	7	**-4**	0	6	4

En este caso el mínimo es -4, que está en la posición 7. Una vez encontrado el mínimo se intercambia en el arreglo los valores de las posiciones 0 y la correspondiente al mínimo:

-4	10	8	1	5	2	7	4	0	6	4

Como es el mínimo sabes que no hay ninguno menor a la derecha de éste. El arreglo está ordenado hasta la posición 0. Pasamos a la posición 1. Ahora encontramos el mínimo de las entradas a partir del índice 1.

-4	10	8	1	5	2	7	4	**0**	6	4

En este caso es 0. Intercambiamos la entrada de la posición 1 con el mínimo:

-4	0	8	1	5	2	7	4	**10**	6	4

A partir de este punto el arreglo está ordenado hasta la posición 1. Pasas a la posición 2 y repites este proceso. Continúa hasta que llegues a la última posición de tu arreglo.

Resumiendo, Selection Sort con una entrada A:

1. Inicia en $i = 0$
2. Para cada índice i en el arreglo A:
   (a) Encuentra el índice del valor mínimo de las entradas desde la posición i (desde $A[i]$ hasta $A[A.length - 1]$).
   (b) Intercambia la entrada de este índice con la entrada de $A[i]$.

4 Desarrollo

Deberás implementar las funciones `selectionSort` y `bucketSort` con los algoritmos correspondientes. Si el algoritmo que implementado no corresponde al nombre de la función no será tomada en cuenta. Además debes implementar la función `intercambia` que intercambia los valores de dos índices de un arreglo.
No modifiques aquellas funciones que no tienen el comentario *"//Implemente la función".*
El método `main` (que se ejecuta con `ant run`)de la práctica está diseñado para que tarde en ejecutarse. Debido a esto se les proporcionan **pruebas unitarias** para verificar que sus algoritmos estén ordenando un arreglo correctamente. Las pueden correr con el comando `ant test`. Es normal que los test fallen si no han terminado su código, pero una vez que hayan implementado un algoritmo, las pruebas correspondientes deben pasar.

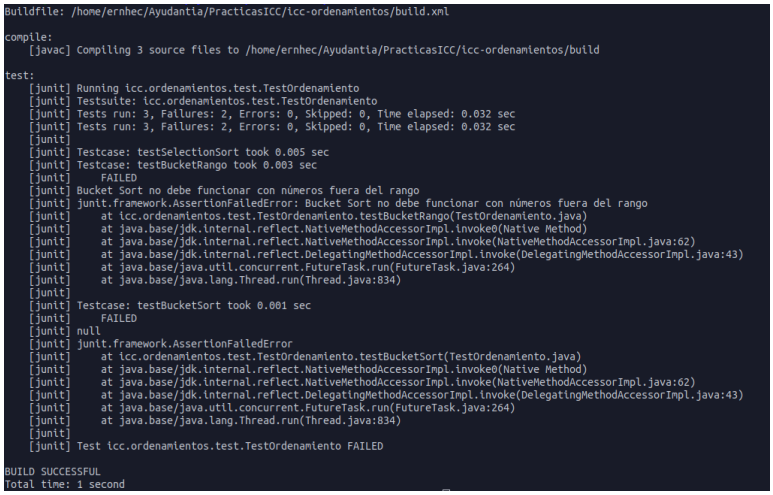

Figure 1: Output esperado de *ant test* si han implementado *selectionSort* pero no *bucketSort*.

5 Entregables

- Deberán entregar el código con las funciones que se les piden implementadas.
- El archivo tiempos.txt que resulta de correr su práctica con `ant run`.
- Las respuestas a las preguntas:
  - Ejecuta la práctica y observa los tiempos de Selection Sort a partir de 50,000 elementos. Elabora una tabla, como la siguiente:

Tamaño	**Factor de Aumento de tamaño respecto al anterior**	**tiempo**	**Factor de aumento en tiempo respecto al anterior**
50,000	-	1000	-
100,000	2 = (50,000/100,000)	4000	4 = (4000/1000)

    ¿Puedes observar **aproximadamente** un comportamiento cuadrático en los tiempos? Entrega también la tabla.
  - ¿Cuanto esperarías que Selection Sort se tardara en tu máquina ordenando un arreglo de 1,000,000 de elementos?