Trabajo Práctico Obligatorio N° 1

Métodos de Ordenamiento y Análisis de Eficiencia

Fecha límite de Entrega: 7/9/2019

Integrantes:

- Morales Matías - FAI 108
- Amarante Carlos - FAI 1922

Algoritmos de ordenamiento asignados: Bubble-Sort, Bucket-Sort, Quick-Sort

Problema Indicado: Suma dígitos
Ordenen una lista de números enteros no por su valor sino por cuanto da la suma de sus dígitos, desempatando en caso de coincidencias por el orden natural de los mismos.

1 – Dados los algoritmos asignados por la cátedra y el problema indicado, se pide:

- Enunciar qué hace cada algoritmo y describir coloquialmente su funcionamiento y características relevantes: ¿Por qué funciona?, ¿Qué determina la performance del algoritmo?, ¿Es un algoritmo estable?, ¿Cuáles son los mejores y peores casos?, ¿Qué orden tiene el espacio de memoria requerido para la ejecución del algoritmo?
- Presentar una implementación funcional de cada algoritmo.
- Realizar el análisis de eficiencia teórico del código presentado: Establecer el Orden, Omega y Theta de ejecución que resulta del análisis anterior.
- Realizar mediciones empíricas de los tiempos de ejecución de cada algoritmo.

¿Son estas mediciones consistentes con el análisis teórico de eficiencia de cada algoritmo?

Bucket Sort

Enunciar qué hace cada algoritmo y describir coloquialmente su funcionamiento y características relevantes: ¿Por qué funciona?

Es un algoritmo de ordenamiento que distribuye todos los elementos a ordenar entre un número finito de casilleros. Cada casillero sólo puede contener los elementos que cumplan unas determinadas condiciones. Las condiciones deben ser excluyentes entre sí, para evitar que un elemento pueda ser clasificado en dos casilleros distintos. Después cada uno de esos casilleros se ordena individualmente con otro algoritmo de ordenación (que podría ser distinto según el casillero), o se aplica recursivamente este algoritmo para obtener casilleros con menos elementos.

<u>El algoritmo contiene los siguientes pasos:</u>

1. Crear una colección de casilleros vacíos
2. Colocar cada elemento a ordenar en un único casillero
3. Ordenar individualmente cada casillero
4. devolver los elementos de cada casillero concatenados por orden

¿Qué determina la performance del algoritmo?

La performance está definida por el rango de los elementos, es decir, que si al aplicar la división de los elementos por rango no son distribuidos de manera uniforme, su efectividad se verá afectada.

¿Cuáles son los mejores y peores casos?

Mejores Casos:

- Los elementos del arreglo se distribuyan de manera uniforme

Peores Casos:

- Todos los elementos del arreglo caigan dentro del mismo bucket, originando que esta implementación pierda su eficiencia, ya que su performance dependerá de que el ordenamiento interno del bucket ordene todos los elementos del arreglo original.

¿Qué orden tiene el espacio de memoria requerido para la ejecución del algoritmo?

Mejor Caso O(n)

```java
public static void bucketsort(int arreglo[], int valorMax, int cantParticiones) {
    // Crea el bucket vacio
    int longitudRango = valorMax / cantParticiones;
    int[][] bucket = new int[cantParticiones][arreglo.length];
    int[] limitesBuckets = new int[cantParticiones];

    // Pone los elementos del arreglo en diferentes buckets
1   for (int i = 0; i < arreglo.length; i++) {
        clasificarNumeros(arreglo[i], bucket, longitudRango, limitesBuckets);
    }

    // ordena los buckets de forma individual
2   for (int i = 0; i < bucket.length; i++) {
        quicksort(bucket[i], 0, limitesBuckets[i] - 1);
    }

    // Concatena los buckets en el arreglo inicial
    int pos = 0;
    for (int i = 0; i < bucket.length; i++) {
        int j = 0;
3       while (j < limitesBuckets[i]) {
            arreglo[pos] = bucket[i][j];
            pos++;
            j++;
        }
    }
}
```

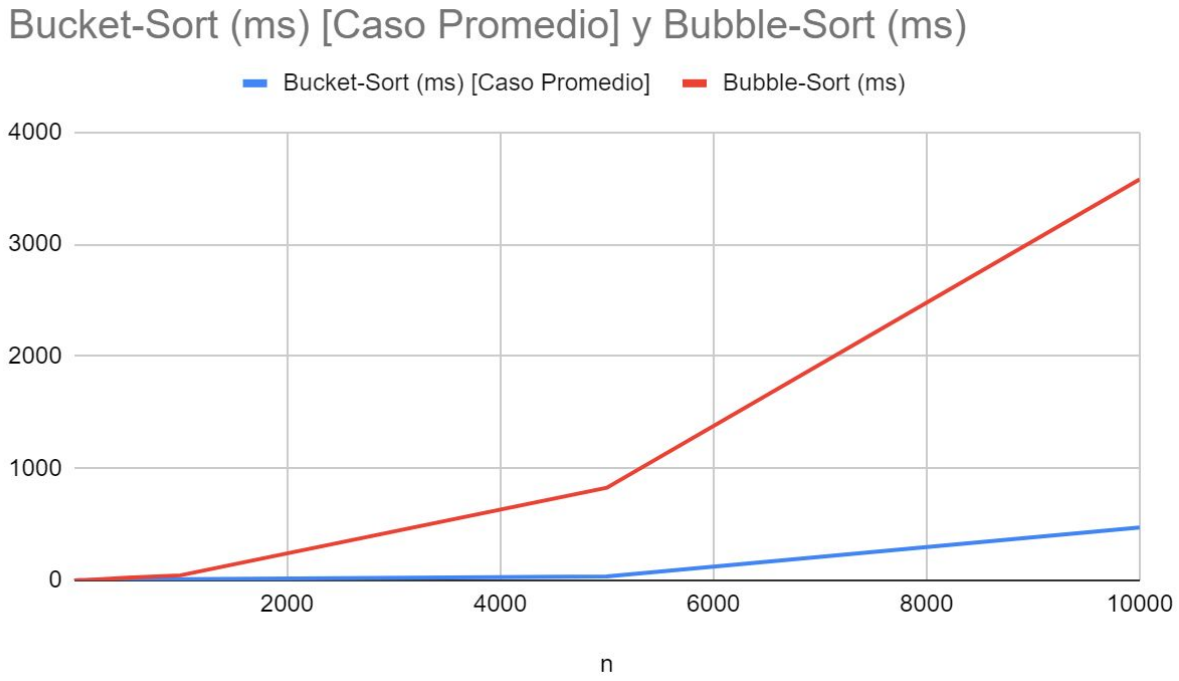
1) El orden del for es O(n) ya que se recorre el arreglo inicial (desordenado) y lo coloca en el bucket adecuado.

2) El orden del for es la es el orden del ordenamiento interno, multiplicado por la cantidad de buckets con elementos.

3) El orden de recorrer el arreglo de buckets es n, ya que se recorre la misma cantidad (n) de elementos del arreglo desordenado.

Datos Empíricos:

n	Bucket-Sort (ms) [Caso Promedio]	Bubble-Sort (ms)
1	0	0
5	0	0
10	0	0
50	1	1
100	1	2
500	6	23
1000	11	45
5000	35	827
10000	471	3579



Quicksort

Enunciar qué hace cada algoritmo y describir coloquialmente su funcionamiento y características relevantes: ¿Por qué funciona?

El algoritmo trabaja de la siguiente forma:

- Elegir un elemento del arreglo de elementos a ordenar, al que llamaremos **pivote**.
- Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
- La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
- Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

¿Qué determina la performance del algoritmo?

Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.

¿Cuáles son los mejores y peores casos?

Mejores Casos:

- En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos sublistas de igual tamaño. En este caso, el orden de complejidad del algoritmo es **O(n·log n)**.

El mejor caso sería si se introdujera los elementos en un árbol AVL y luego se obtendría un array o lista según su recorrido inorden

Peores Casos:

- En el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de **O(n²)**. El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas. Pero principalmente depende del pivote, si por ejemplo el algoritmo implementado toma como pivote siempre el primer elemento del array, y el array que le pasamos está ordenado, siempre va a generar a su izquierda un array vacío, lo que es ineficiente.

¿Qué orden tiene el espacio de memoria requerido para la ejecución del algoritmo?

En el caso promedio, el orden es **O(n·log n)**.

Peor caso $\Omega(n^2)$

Mejor caso Θ

```
public static void quicksort(int[] arreglo) {
    if (arreglo != null) {
        quicksort(arreglo, 0, arreglo.length - 1);
    }
}

private static void quicksort(int[] arreglo, int i, int j) {
    // i posiciÃ³n inicial
    // j posicion final
    int indiceP;
    if (i < j) {
        indiceP = particion(arreglo, i, j);
        quicksort(arreglo, i, indiceP - 1);
        quicksort(arreglo, indiceP + 1, j);
    }
}
```

Como se puede ver, el orden del quicksort utiliza un algoritmo recursivo. El orden de los algoritmos recursivos varía desde **n** (si la función recursiva es llamada n veces) hasta $\mathbf{2^n}$ (si la función recursiva es llamada 2 veces en cada recursión).

Como el algoritmo quicksort utilizado se utiliza en la forma de algoritmo “divide y vencerás” , es decir que en cada recursión se analiza la mitad de los elementos. Por lo tanto el orden es n.log n .

```java
private static int particion(int[] arreglo, int min, int max) {
    int i, pivote;
    pivote = arreglo[max];
    i = (min - 1); // indice del elemento mas chico

    for (int j = min; j < max; j++) {
        // Si el elemento actual es menor que el pivote
        if (SumaDigitos.mayorQue(pivote, arreglo[j])) {
            // if (arreglo[j] < pivote) {
            i++;

            // intercambia arreglo[i] y arreglo [j]
            permuta(arreglo, i, j);
        }
    }
    permuta(arreglo, i + 1, max);
    return i + 1;
}
```

El algoritmo de partición tiene un orden n, ya que recorre todo el arreglo (en la primer vuelta) y ubica al pivote en la posición correcta. En el peor de los casos, que el arreglo ya esté ordenado, se recorrerá todo el arreglo sin hacer ningún cambio, intercambiando el mismo elemento, por el mismo elemento.

2 – Resolver el problema propuesto a cada grupo, usando de entre los Algoritmos asignados al grupo, el que resulte más apropiado. Justificar la selección.

En el caso de que en el peor caso del Bucket sort, caigan todos los elementos del arreglo en un solo bucket, se procede a ordenar los elementos internos del bucket mediante el método quicksort, que tiene un orden mucho menor al método de ordenamiento burbuja, por lo tanto, aun ante el peor caso del bucket sort, tendríamos un mejor tiempo de ejecución.

3 – Presentar el informe con los apellidos y nombres de cada integrante.