Taller #2: Evaluación Experimental de Algoritmos

Profesor:	Jorge M. Londoño
Estudiantes	Julian Esteban Gómez López - Julian Styven Colorado Agudelo
IDs	000438304 - 000420606

Enunciado:

En está práctica se implementará el ADT Conjunto utilizando listas enlazadas (e.g. Bag, LinkedList) y se implementan algunas de las principales operaciones de conjuntos. Interesa estimar el tiempo requerido por algunos de estos algoritmos en forma experimental y/o analítica.

Ejercicios a desarrollar:

a. Implemente un ADT Conjunto que represente un conjunto de elementos de un tipo genérico T. Siguiendo la definición matemática de conjunto, no se permiten elementos repetidos. Implementar las siguientes operaciones como parte del API del ADT Conjunto: **Agregar un elemento (add)**, **eliminar un elemento (remove)** y **determinar si un elemento pertenece al conjunto (in)**, **cardinalidad del conjunto**
(size) y **determinar si es** el **conjunto vacío (isEmpty)**. Observar que la operación add no debe permitir elementos repetidos en el conjunto.

b. Completar la siguiente tabla indicando el orden de crecimiento (estimado de forma analítica) de cada operación según su implementación:

Operación	Orden de Crecimiento
add	n^2
remove	n^2
in	n
size	n
isEmpty	a (constante)

c. Hacer un método estático que calcule el hashDelEquipo. Para esto, se concatenan los String correspondientes al ID de cada miembro y se obtiene el hashCode() del String resultante. Calcular también el resultado del hashCode() módulo 4 para este String. La siguiente es la firma de la función hashDelEquipo:

   int hash = ((id1+id2).hashCode())%4; //Hash del equipo % 4

   int hashEquipo(String id1, String id2)

d. De acuerdo con el valor del hashEquipo obtenido en el numeral anterior, implementar la operación de conjuntos correspondiente según la siguiente tabla:

hashCode(IDs)	Algoritmo
0	union
1	intersection
2	difference
3	cartesianProduct

e. Determinar analíticamente el orden de crecimiento para la operación seleccionada.

f. Implementar una función de biblioteca que evalúe experimentalmente el tiempo de ejecución de la operación seleccionada, siguiente el siguiente procedimiento:

   a. Se itera para distintos valores del tamaño de los conjuntos, utilizar una estrategia de doblado del tamaño de la entrada, por ejemplo: N=1000, 2000, …, 64000, etc.

   b. Para cada valor de N se hacen k=20 pruebas de ejecución del algoritmo. En cada prueba se genera un dos conjuntos de N elementos, se registra el tiempo de ejecución del algoritmo asignado y se reporta el tiempo de ejecución para cada prueba y cada N.

      Nota: Tener presente no incluir la generación del conjunto, ni hacer operaciones de entrada/salida durante la medición del tiempo.

   c. El método main invoca esta función para ejecutar toda la prueba.

g. Los datos obtenidos de la evaluación del algoritmo se tabulan en una hoja de cálculo. Para cada valor de N calcular el mínimo, máximo y promedio de los tiempos de ejecución.

h. Hacer una gráfica de los tiempos de ejecución mínimo, máximo y promedio como función de N.

i. Obtener la curva de mejor ajuste para las tres curvas obtenidas e indicar la expresión que mejor se ajusta a la curva utilizando el método de interpolación. Indicar la expresión obtenida.

j. Comparar la expresión obtenida de las curvas de mejor ajuste con el orden de crecimiento del algoritmo correspondiente obtenido en el numeral (e).

Enlace a Github:

Para la solución del taller, creamos un repositorio en Github y procedimos a crear el proyecto usando NetBeans en el cuál está el código fuente de la solución implementada, en el mismo repositorio está el archivo de Excel, el procedimiento con el cuál se calcularon los órdenes de crecimiento de las operaciones solicitadas.

El método Main está en la clase llamada “EvaluacionExperimentalAlgoritmos”

Clic para ir al repositorio

Operación Seleccionada: UNION

```
Union entre conjuntos
*/
public static Conjunto Union(Conjunto grupo1, Conjunto grupo2){
    Conjunto resultado = grupo1; //Se empieza asi para luego ir añadiendo los elementos del otro conjunto

    int nGrupo2 = grupo2.Size();
    //Se recorre el segundo conjunto y se van sacando sus elementos para pasarlos al resultado
    for(int i = 0; i<nGrupo2;i++){
        resultado.Add(grupo2.Remove());
    }
    return resultado;
}
```

Orden de crecimiento de la operación Unión: N

Resultado del análisis

> 💡 La impresión del resultado de cada unión está comentada por defecto para optimización del código, los análisis se hicieron con esta impresión desactivada.

Cantidad Conjuntos	Tiempo
1000	20
2000	104
4000	294
8000	1063
16000	9716
32000	14497
64000	63791
128000	255758
256000	1068196

Para analizar de mejor forma el algoritmo no solo se obtiene el tiempo de cada ejecución si no de todo por completo.
Cada ejecución tarda 0 milisegundos independiente del tamaño del conjunto, aunque la operación completa sí tarda un tiempo más considerable

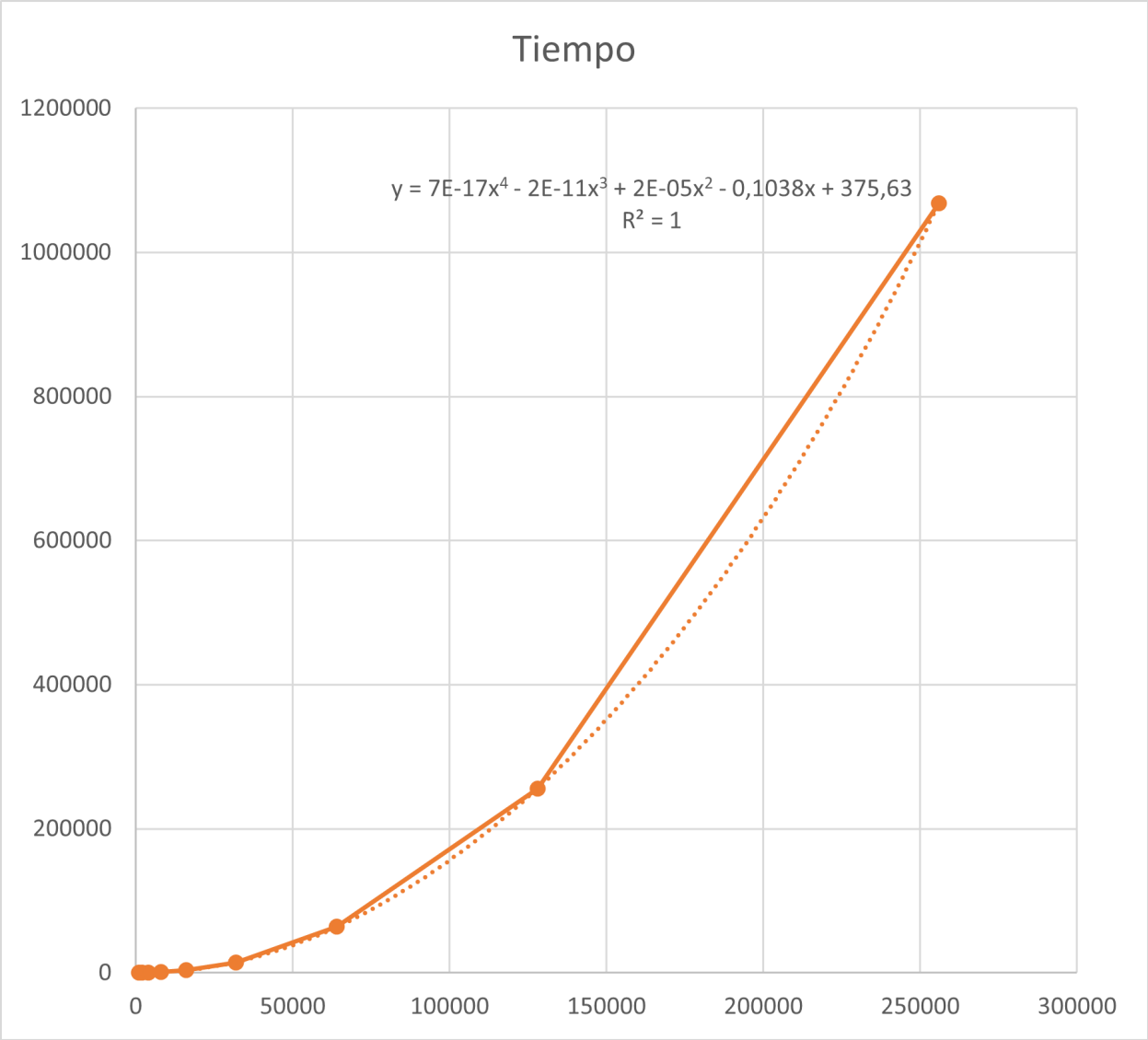
```
La union numero 17 tardo: 0
La union numero 18 tardo: 0
La union numero 19 tardo: 0
La union numero 20 tardo: 0
El tiempo que se usó para realizar todas las uniones es de: 1062
Se analizaron 8000 elementos
Introduzca cualquier cosa para duplicar
o introduzca N para detener
```

En vista de que cada unión no tiene un tiempo considerable, en el excel y en la gráfica analizaremos cada tiempo de ejecución por completo

Elementos	Tiempo
1000	20
2000	104
4000	294
8000	1063
16000	3716
32000	14497
64000	63791
128000	255758
256000	1068196

Tabla obtenida

Y el grafico resultante es:



Donde la curva que más se adapta es una polinómica de grado 4