Hoja de Trabajo 9

Boggdan Barrientos 14484
Andre Rodas 14395
Rudy Garrido 14366

Pruebas unitarias:

SimpleSet (Clase Original):

Call Tree - Method	Total Time [%]	Total Time	Invocations
main		16,453 ms (100%)	1
SimpleSet.**get** (Word)		16,461 ms (100%)	6606
SimpleSet.**add** (Word)		19.3 ms (0.1%)	68429
SimpleSet.**<init>** ()		0.206 ms (0%)	1

RedBlackTreeSet:

Call Tree - Method	Total Time [%]	Total Time	Invocations
main		3,358 ms (100%)	1
RedBlackTreeSet.**add** (Word)		3,377 ms (100%)	66032
RedBlackBST.**put** (Comparable, Object)		3,347 ms (99.7%)	66032
RedBlackBST.**put** (RedBlackBST.Node, Comparable, Object)		3,298 ms (98.2%)	66032
Self time		39.1 ms (1.2%)	66032
RedBlackBST$Node.**access$002** (RedBlackBST.Node, boolean)		10.3 ms (0.3%)	66031
Self time		29.4 ms (0.9%)	66032
RedBlackTreeSet.**<init>** ()		5.15 ms (0.2%)	1

SplayTreeSet:

Call Tree - Method	Total Time [%]	Total Time	Invocations
main		653 ms (100%)	1
SplayTreeSet.**add** (Word)		664 ms (100%)	47865
SplayTree.**insert** (Word)		634 ms (97.1%)	47865
SplayTree.**splay** (Word)		535 ms (81.9%)	47864
Self time		72.4 ms (11.1%)	47865
Word.**compareTo** (Word)		16.4 ms (2.5%)	47863
BinaryNode.**<init>** (Word)		10.5 ms (1.6%)	41316
Self time		29.4 ms (4.5%)	47865
SplayTreeSet.**<init>** ()		7.20 ms (1.1%)	1

HashTableSet

Call Tree - Method	Total Time [%]	Total Time	Invocations
main		87.7 ms (100%)	1
HashtableSet.**add** (Word)		107 ms (100%)	68429
HashtableSet.**get** (Word)		7.59 ms (8.7%)	6606
HashtableSet.**<init>** ()		0.140 ms (0.2%)	1

TreeMapSet

Call Tree - Method	Total Time [%]	Total Time	Invocations
main		242 ms (100%)	1
TreeMapSet.**add** (Word)		247 ms (100%)	68429
TreeMapSet.**get** (Word)		21.8 ms (9%)	6606
TreeMapSet.**<init>** ()		0.340 ms (0.1%)	1

CLASE PROPUESTA: **HashTableSet**

R: Luego de analizar cada clase, se llega a la conclusión que los métodos recursivos en este caso consumen mayor tiempo de ejecución, por lo que, el menor tiempo lo posee el HashTableSet, las razones pueden ser: posee una complejidad basada en una constante y no un logaritmo. Al ser palabras es ideal para insertar y buscar, debido a que se posee un buen hashing y 'relativamente' pocos elementos, por lo que no hay colisiones y es más fácil buscar en insertar.