OPERACIONES DEL TAD SORTEDSET

<table>
<tr><td>Operación</td><td>Agregar</td></tr>
<tr><td>Precondicion</td><td>Ninguna, se garantiza esta precondición al no haber precondición para el agregar del ABB utilizado</td></tr>
<tr><td>Poscondicion</td><td>Se ha agregado un nuevo elemento al conjunto ordenado, se garantiza esta condición al usar un ABB puesto que el ABB agrega ordenadamente elementos.</td></tr>
<tr><td>Salida</td><td>Ninguna</td></tr>
<tr><td>Errores</td><td>Se lanza exception al tratar de agregar un elemento ya existente en el conjunto, se garantiza este error al buscar si el elemento esta en el árbol antes de agregarlo</td></tr>
<tr><td>Complejidad</td><td>
<table>
<tr><td>Complejidad</td><td>PseudoCodigo</td></tr>
<tr><td>n</td><td>If buscar!=null then</td></tr>
<tr><td>1</td><td>Error</td></tr>
<tr><td>0</td><td>else</td></tr>
<tr><td>n</td><td>arbol . agregar(x,x)</td></tr>
<tr><td>Total: O(n)</td><td></td></tr>
</table>
La búsqueda se da en O(n) en el peor de los casos, el cual es el árbol que tiene una única rama como si fuera una lista. El agregar también es O(n) puesto que en el peor de los casos se deberá recorrer todos los elementos del arbol para agregar uno al final. En total la complejidad del algoritmo es O(n).</td></tr>
</table>

<table>
<tr><td>Operación</td><td>eliminar</td></tr>
<tr><td>Precondicion</td><td>Ninguna, se garantiza precondición al no necesitarse ninguna precondición tampoco para el ABB utilizado en la operación eliminar</td></tr>
<tr><td>Poscondicion</td><td>Se ha eliminado un elemento del cojunto pero este aun sigue siendo ordenado, se garantiza esta precondición con el ABB puesto que el ABB una vez eliminado un elemento conserva su propiedad de orden</td></tr>
<tr><td>Salida</td><td>Ninguna</td></tr>
<tr><td>Errores</td><td>Se lanza exception al tratar de eliminar un elemento que no existe en el conjunto, se garantiza este error al verificar primero si el elemento esta en el ABB por medio de una búsqueda y también se tira exception si se comprueba que el conjunto es vacio y se trata de eliminar un elemento</td></tr>
<tr><td>Complejidad</td><td>
<table>
<tr><td>Complejidad</td><td>PseudoCodigo</td></tr>
<tr><td>n</td><td>If buscar!=null ó raíz = null then</td></tr>
<tr><td>1</td><td>Error</td></tr>
<tr><td>0</td><td>else</td></tr>
<tr><td>n</td><td>arbol . eliminar(x)</td></tr>
<tr><td>Total: O(n)</td><td></td></tr>
</table>
La búsqueda se da en O(n) en el peor de los casos, el cual es el árbol ABB que tiene una única rama como si fuera una lista. La remoción con el ABB se da en O(n) en el peor de los casos el cual es en el que el árbol tiene una única rama como si fuera una lista. En total el algoritmo tiene O(n)</td></tr>
</table>

<table>
<tr><td>Operación</td><td>Pertenece</td></tr>
<tr><td>Precondicion</td><td>Ninguna</td></tr>
<tr><td>Poscondicion</td><td>Ninguna, no se ha modificado el estado del conjunto</td></tr>
<tr><td>Salida</td><td>Boolean, true si el elemento existe en el conjunto, false si el conjunto es vacio o si no existe el elemento. Se garantiza por medio de la búsqueda del ABB .</td></tr>
<tr><td>Errores</td><td>Ninguno</td></tr>
<tr><td>Complejidad</td><td>
<table>
<tr><td>Complejidad</td><td>PseudoCodigo</td></tr>
<tr><td>1</td><td>If raíz != null then</td></tr>
<tr><td>n</td><td>buscar (x)</td></tr>
<tr><td>Total: O(n)</td><td></td></tr>
</table>

O(n) puesto que en un ABB, la búsqueda es O(n) en el peor caso y Omega de logn en el mejor caso, esto garantiza la rápida localización de los elementos para cumplir con la función de rápida verificación de elementos en el conjunto para lo cual se creó el SortedSet.
</td></tr>
</table>

<table>
<tr><td>Operación</td><td>Mostrar</td></tr>
<tr><td>Precondicion</td><td>Ninguna</td></tr>
<tr><td>Poscondicion</td><td>Ninguna, no se ha modificado el estado del conjunto</td></tr>
<tr><td>Salida</td><td>String, un string con el valor de todos los elementos del conjunto, se garantiza por medio del inorden de ABB que se encarga de recorrer el árbol conctruyendo un string con el valor de todos los elementos en el ABB en un orden de menor a mayor según su llave</td></tr>
<tr><td>Errores</td><td>Ninguno</td></tr>
<tr><td>Complejidad</td><td>
<table>
<tr><td>Complejidad</td><td>PseudoCodigo</td></tr>
<tr><td>n</td><td>arbol.inorden()</td></tr>
<tr><td>Total: Θ (n)</td><td></td></tr>
</table>

Θ (n) puesto que en un ABB y en los arboles en general, los recorridos toman igual número de operaciones que los elementos del conjunto.
</td></tr>
</table>

<table>
<tr><td>Operación</td><td>Interseccion</td></tr>
<tr><td>Precondicion</td><td>Ninguna</td></tr>
<tr><td>Poscondicion</td><td>Ninguna, no se ha modificado el estado del conjunto</td></tr>
<tr><td>Salida</td><td>SortedSet interseccion con el SortedSet pasado por parámetro, se garantiza que el sorted set salida no tendrá ni elementos repetidos y todos estarán en orden, puesto que por medio de isElement se comprueba la existencia en común de todos los elementos.</td></tr>
<tr><td>Errores</td><td>Ninguno</td></tr>
<tr><td>Complejidad</td><td>
<table>
<tr><td>Complejidad</td><td>Pseudocodigo</td></tr>
<tr><td>1</td><td>Conj2 <- new conjunto</td></tr>
<tr><td>n^2</td><td>For i<-0 to conj1.longitud do</td></tr>
<tr><td>n</td><td>If s2.isElement(conj1.get(i)) then</td></tr>
<tr><td>n</td><td>Conj2.add(conj1.get(i))</td></tr>
<tr><td>Total: n^2</td><td></td></tr>
</table>
S2 es el sorted pasado por parametro
Se ha hace un ciclo externo por el peso del árbol, el cual es n y adentro se comprueba la existencia del elemento dentro del sorted set pasado por parámetro S2 con el método isElement() ,el cual también tiene una complejidad de n , al ser dos n anidados la complejidad total es O(n^2)
</td></tr>
</table>

<table>
<tr><td>Operación</td><td>Union</td></tr>
<tr><td>Precondicion</td><td>Ninguna</td></tr>
<tr><td>Poscondicion</td><td>Ninguna, no se ha modificado el estado del conjunto</td></tr>
<tr><td>Salida</td><td>SortedSet unión con el SortedSet pasado por parámetro, se garantiza que el sorted set salida no tendrá ni elementos repetidos y todos estarán en orden. Puesto que al ser un sortedset un árbol organizado el proceso de agregar los elementos garantiza que ellos quedaran en orden.</td></tr>
<tr><td>Errores</td><td>Ninguno</td></tr>
<tr><td>Complejidad</td><td>
<table>
<tr><td>Complejidad</td><td>Pseudocodigo</td></tr>
<tr><td>n</td><td>copiaConj2 <-s2.clone()</td></tr>
<tr><td>n^2</td><td>for i<-0 to conj1.longuitud do</td></tr>
<tr><td>n</td><td>if (copiaConj2.isElment(conj1.get(i))==false) then</td></tr>
<tr><td>n</td><td>copiaConj2.add(conj1.get(i))</td></tr>
<tr><td>Total:O(n^2)</td><td></td></tr>
</table>
S2 es el sorted pasado por parametro
El cilo externo aporta complejidad n, y tanto isElement() como add(), por estar implementados con el arbol ABB son n también. Estos dos n anidados dan como complejidad O(n^2)
</td></tr>
</table>