TimSortError

Sometimes, simple things like sorting are not simple. We were doing a sorting on a large data set containing around 2000 elements. Sometimes, when most of these elements had null fields, we would get a very weird exception:

java.lang.IllegalArgumentException: Comparison method violates its general contract!

at java.util.TimSort.mergeLo(TimSort.java:773)

at java.util.TimSort.mergeAt(TimSort.java:510)

at java.util.TimSort.mergeCollapse(TimSort.java:437)

at java.util.TimSort.sort(TimSort.java:241)

at java.util.Arrays.sort(Arrays.java:1512)

at java.util.ArrayList.sort(ArrayList.java:1454)

at java.util.Collections.sort(Collections.java:175)

Did a bit of googling, and found out this happens when the sorting implementation is not transitive.

The problem that we faced is really, reproducing this consistently, as it seemed to work in most cases and then starts failing seemingly randomly.

In this entry, I will try to reproduce this issue. Let us consider the below entity:

**public** **class** Person {

**private** **final** String name;

**private** **final** Integer id;

**public** Person(String name, Integer id) {

**this**.name = name;

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** Integer getId() {

**return** id;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + ((id == **null**) ? 0 : id.hashCode());

result = prime \* result + ((name == **null**) ? 0 : name.hashCode());

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Person other = (Person) obj;

**if** (id == **null**) {

**if** (other.id != **null**)

**return** **false**;

} **else** **if** (!id.equals(other.id))

**return** **false**;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**return** **true**;

}

@Override

**public** String toString() {

**return** "Person [name=" + name + ", id=" + id + "]";

}

}

We would use the below comparator for sorting this:

**public** **class** PersonComparatorNonTransitive **implements** Comparator<Person> {

@Override

**public** **int** compare(Person person1, Person person2) {

**if** (person1.getId() == **null**) {

**return** -1;

}

**if** (person1.getId() != **null**) {

**return** 1;

}

**if** ((person1.getId() == **null**) && (person2.getId() == **null**)) {

**return** 0;

}

**return** person1.getId() - person2.getId();

}

}

We will try to sort a List<Person> with the above comparator:

List<Person> persons = ...;

Collections.*sort*(persons, **new** PersonComparatorNonTransitive());

This would fail consistently with the below data set:

**private** List<Person> getDataSetForTimSortException() {

List<Person> persons = **new** ArrayList<>();

persons.add(**new** Person("TestName", 3));

persons.add(**new** Person("TestName", 290));

persons.add(**new** Person("TestName", 1));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", **null**));

persons.add(**new** Person("TestName", 1));

persons.add(**new** Person("TestName", 398));

persons.add(**new** Person("TestName", 46));

persons.add(**new** Person("TestName", 45));

persons.add(**new** Person("TestName", 0));

persons.add(**new** Person("TestName", 3));

persons.add(**new** Person("TestName", 45));

persons.add(**new** Person("TestName", 130));

persons.add(**new** Person("TestName", 33));

persons.add(**new** Person("TestName", 56));

**return** persons;

}

The full test case is here:

<https://github.com/paawak/blog/blob/master/code/sort/src/test/java/com/swayam/demo/sort/PersonComparatorNonTransitiveTest.java>

Now, consider the below comparator:

**public** **class** PersonComparatorTransitive **implements** Comparator<Person> {

@Override

**public** **int** compare(Person person1, Person person2) {

**if** ((person1.getId() == **null**) && (person2.getId() != **null**)) {

**return** -1;

}

**if** ((person1.getId() != **null**) && (person2.getId() == **null**)) {

**return** 1;

}

**if** ((person1.getId() == **null**) && (person2.getId() == **null**)) {

**return** 0;

}

**return** person1.getId() - person2.getId();

}

}

The difference between the transitive and the non-transitive version is obvious. The real challenge here is having a data-set that fails consistently to cause the exception in TimSort.

The entire source can be found here:

<https://github.com/paawak/blog/tree/master/code/sort>