③ return I2.compareTo(I1);

↳ [descending order] → [20, 15, 10, 5, 0]

④ return -I2.compareTo(I1);

↳ Ascending order [0, 5, 10, 15, 20]

⑤ return +1;

(> [Insertion order) → [10, 0, 15, 5, [illegible], 20, 20]

⑥ return -1; (Reverse of insertion order) ⇒ [20, 20, 5, 15, 0, 10]

⑦ return 0; [only first element will be inserted & all remaining will be considered as duplicate.] → [10].

⇒ WAP to insert String objects into the TreeSet where all elements should be inserted a/c to reverse of Alphabetical order.

Sol^n:-

```
import java.util.*;
public class Test {
    p s v m(String[] args){
TreeSet t = new TreeSet(new MyComparator());
        t.add("Rosa"); t.add("SubhaRani");
t.add("Rajakumari"); t.add("GangaBhavani"); t.add("Ramulamma");
        s.o.p(t);
}}
class MyComparator extends implements Comparator
{
    public int compare(Object obj1, Object obj2)
    {
        String s1 = obj1.toString();
        String s2 = (String) obj2;
        return s2.compareTo(s1);
        // retur -s1.compareTo(s2);
}}
```

=> WAP to insert StringBuffer objects into the TreeSet where sorting order is Alphabetical order.

Sol:-

```
import java.util.*;
public class Test
{
    p s v m(String[] args)
{
TreeSet t = new TreeSet(new MyComparator());
   t.add(new StringBuffer("A");
   t.add(new StringBuffer("Z");
   - - - - - - - - ("K");
   - - - - - - - - ("L");
   S.O.p(t);
}}
}

Class MyComparator implements comparator
   {
   public int compare(Object obj1, Object obj2)
   {
   String s1 = obj1.toString();
   String s2 = obj2.toString();
      return s1.compareTo(s2);
}  }
```

Note:- If we are ~~defining~~ depending on default natural sorting order, compulsary objects should be homogenious and comparable otherwise we will get R.T. Exception saying class cast Exception.

If we are defining our own sorting by comparator, then objects need not be comparable and homogenious. i.e. Heterogenious, Non comparable objects also.