1. Experiment

One way to sort arrays is by using the algorithm *insertion sort.* The algorithm can be used for both strings and integers.

The task is to find out how many strings and integers can be sorted in 1 second using this algorithm. The arrays to be sorted will be created with random generated integers and strings with the length of 10 characters. To generate strings, a private method will be created. In order to reduce the number of duplicate elements, the range used by the random generator will be larger than the array-size. Each test was executed 10 times to get an average result.

2. Experiment Setup

All experiments were done on a Lenovo ThinkPad t460s with an Intel Core i5 processor (2.50Ghz) with 12GB of memory. To avoid memory problems, garbage collection was also run just before and after each test using *System.gc()*. All other applications were closed while performing the experiments. The clock *System.currentMillis()* was used in all tests.

To make sure that the JVM is fully optimized when we do the actual tests, we go through with a few practice tests to ensure accurate results.

The idea to make the test is to create an array with an optional size, generate random integers and strings and put them in the array, as many as the size of the array is. Then we can move on to make the test. I saved the current time in a variable, made a for-loop with the termination equal to the size of the array. I stored the element from the array in position what the current loop counter value is. Declared a variable j to the value of the loop counter value decreased by one.

Then I did a while-loop with the condition: *j >= 0 && array[j]>storage.* This means that it will loop while j is bigger or equal to 0 and the value in the array at position j is bigger than the value we have stored. While this condition is true, we loop through the array and move all elements the left and decrease j every time. When we have reached a smaller element compared to the stored, the loop breaks, and we insert the stored element at the current position. When we have looped through the whole array, we save the current time in another variable, calculate the total time for the test and print it out. We can change the size of the array, so that we get a total time as near as 1 second as possible. As an example, the code in the image below shows the code used for measuring the time to sort an array with integers:

```java
int[] randomInts = new int[80100]; // create an array with the size of 80 100

Random rand = new Random();
for (int i = 0; i <randomInts.length ; i++) { // generate random ints and add them to array
    randomInts[i] = rand.nextInt( bound: 100000000);
}
System.gc();
long beforeInt = System.currentTimeMillis(); // we save current time (before test execution)
for (int i = 1; i < randomInts.length; i++) { // for each element in the array
    int storage = randomInts[i]; // we store the current value
    int j = i - 1;
    while (j >= 0 && randomInts[j] > storage) { // while j is bigger or equal to 0 and the element at pos j
        randomInts[j + 1] = randomInts[j];      // is larger than the stored, we move the elements to the left
        j--;    // decrease j.
    }
    randomInts[j + 1] = storage; // when the while-loop is done, we put the stored element at the empty pos.
}
long afterInt = System.currentTimeMillis(); // we save current time (after test execution)
System.gc();
long totalInt = afterInt - beforeInt; // calculate the total time it took to execute the test
```

To generate random strings, the method shown in the image below was created. A string with characters was made. With a for-loop, that loops 10 times, using StringBuilder random characters, from the string *characters*, was appended.

```java
private static String generateString(){
    String characters = "ABCDEFGHIJKLMNOPQRSTUVXYZ";
    StringBuilder text = new StringBuilder();
    Random rand = new Random();
    for (int i = 0; i <10 ; i++) {
        text.append(characters.charAt(rand.nextInt(characters.length())));
    }
    return text.toString();
}
```

3. Experimental Results

3.1 Experiment with random generated integers

The table below shows the time in milliseconds for each test. Every test sorted an array with the 80 100 randomly generated integers. The average time to sort 80 100 randomly generated integers was 934 milliseconds.

Test	Time
1	968
2	981
3	931
4	982
5	992
6	955
7	906
8	952
9	908
10	1064
Average time:	934

3.2 Experiment with random generated strings

The table below shows the time in milliseconds for each test. Every test sorted an array with the 22 000 randomly generated strings. The average time to sort 22 000 randomly generated strings was 1032 milliseconds.

Test	Time
1	1138
2	1008
3	1037
4	952
5	886
6	1061
7	959
7	992
9	1104
10	1182
Average time:	1032

4. Discussion

Clearly, using the insertion sort algorithm, sorting integers outperforms sorting strings. Using the insertion sort, you can sort 58 100 more integers than strings during 1 second. I think the big difference depends on that when you sort strings, you must compare each character in the string.

A problem when testing the algorithm to sort strings, was that depending on what kind of strings were generated, it took less or more time. For example. If two generated strings looked like this: XOUSYDLEOE and XOUSYDLEOY. You must compare every character, because they are equal except the last character and sort them according to the last character.

This made it difficult to get a total time close to 1 second, but as you can see in the result, the average time to sort 22 000 strings took 1032 milliseconds, that is close to 1 second.