In this exercise we study the problem of searching elements from collection (large group of elements).

Execution time measurement system

Java class Stopwatch has been developed to measure a given method's execution time. There are a couple of methods available in the class Stopwatch, but the most important method is

```
      public void measure(Test method)
```

which is given a reference to a class who implements an interface Test. This interface is defined in Stopwatch to be as

```
    @FunctionalInterface
    public interface Test {
        void test();
    }
```

i.e. the interface has one method test(), which is the method whose execution time will be measured by calling the measure method in Stopwatch class[1]. Result of the measurement is given by calling the method

```
      public long toValue()
```

which returns the result in nanoseconds, or

```
      void toValue(StringBuilder result)
```

which append the result (in ms) to the given string.
There is also a file MaxSubSeq.java available which shows how to use the execution measurement system in practice.

Excercise 1a. (Implement and test sequential and binary search algorithms, a-b together 1p)

Implement a class who is able to store elements and search for a given element with find() -type method. The algorithm used for searching is a sequential search algorithm.

```
      public class MySearch {
          private Comparable[] array;

          void setArray(Comparable[] array);
          int LinearSearch(Comparable elem);
          int BinarySearch(Comparable elem);
      }
```

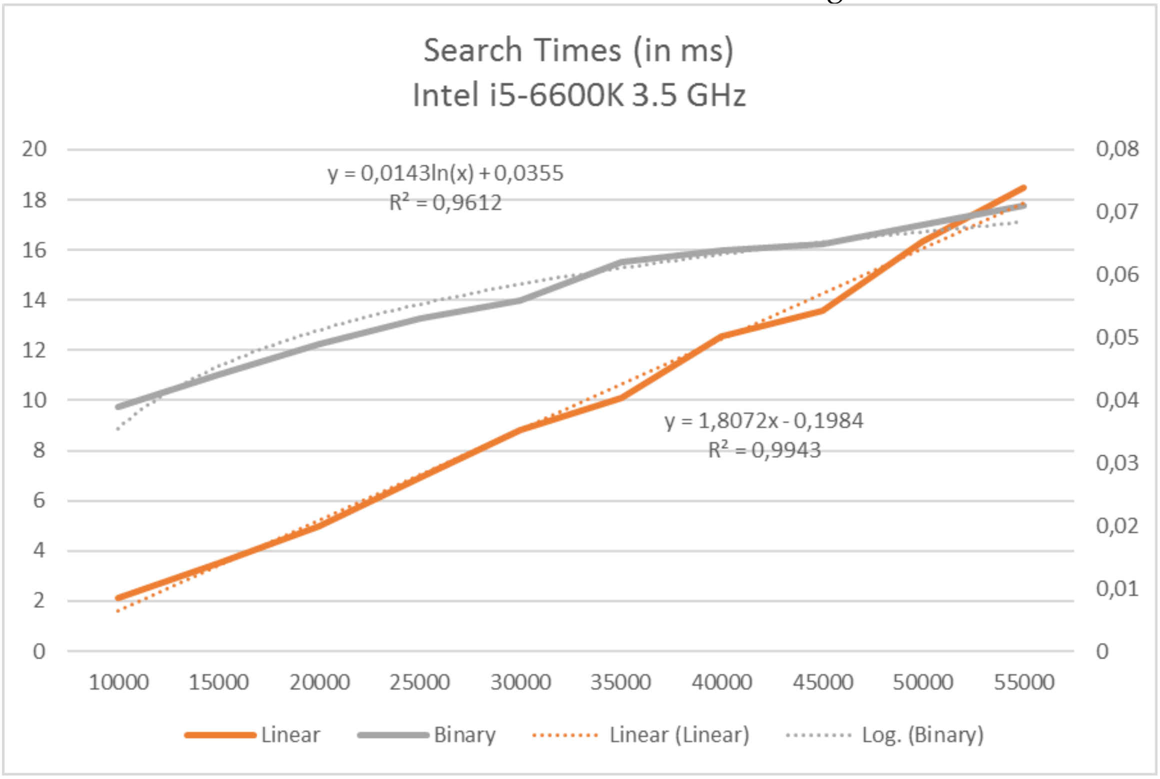
LinearSearch() and BinarySearch() returns the index of the element if the element is found in the array, -1 otherwise. Notice that we use Comparable object, so that we have compareTo() method to compare between two elements.

Exercise 1b. (Measure and compare performances of implemented search algorithms)

Use the given program execution time measurement software package (Stopwatch.java) to measure the execution speed of the sequential and binary search algorithms when the element is an Integer (which implements the Comparable interface). Measure the

[1] This measure() method then calls test() method multiple times and measures the execution time using System.nanoTime() method.

execution speeds when the size of the collection (with random string entries) varies from 10000 elements to 55000 elements in step of 5000 elements. Because the search process itself is quite a rapid operation, repeat the search 500 times in test() method. In order to minimize the effect of a single element to be searched, search different element each time (i.e. create a list of 500 random elements to be searched for). Do 24 experiments per one collection size with random element value (which should be found in the collection) and calculate the median of the results[2]. You should have something like this



From the picture you can verify that the sequential search has O(N) timing requirements (there is only 0,57% chance that the fitness of the linear trend is due to random variations in the data), and binary search has O(logN) requirements[3].

Exercise 1c. (Extra exercise, measure and compare performances of implemented search algorithms when element is String, 0,25p)

Change the element from Integer to random String of 32 character length. Measure the execution times of linear and binary search algorithms. How must slower the String searching process is compared to the Integer searching process?

[2] This is taken care in Stopwatch class
[3] Timing measurement is more unreliable when the times in general are small, therefore binary search curve fitting is not so good (3,88% change that the fitness of the linear trend is due to random variations in the data)