Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Activity 1. [Calculate how many more years we can continue using this way of counting.]

The biggest number we can represent with long is 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 in binary, which are 292278994 translated into decimal. This means the maximum number of years we can represent is 292278994. Taking into account that we have started counting since 1-1-1970, which means that have passed 54 years until today, we can keep using this way of counting for 292278940 years.

Activity 2. [What does it mean that the time measured is 0?

From what size of problem (n) do we start to get reliable times]

It means that the measured time is lower than 50 milliseconds, which is not representative.

We start getting reliable times when measured times are equal or bigger than 50ms, in my case, it is with a size of n = 14000000

Activity 3. [What happens with the time if the problem size is multiplied by 2?

What happens with the time if the problem size is multiplied by a value k other than 2? (try it, for example, for k=3 and k=4 and check the times obtained)

From what we saw in Vector4.java measuring the times for sum, create the following three java classes:

• Vector5.java to measure times for maximum.
• Vector6.java to measure times for matches1.
• Vector7.java to measure times for matches2.

Indicate the main features (processor and memory) of the computer where times have been measured.

Once both tables are filled in, conclude whether the times obtained meet what was expected,
given the computational time complexity of the different operations.]

1- As we are in a linear complexity, times are also multiplied by 2

Base times:

```
SIZE=10 TIME=79 milliseconds SUM=-312 NTIMES=1000000
SIZE=50 TIME=218 milliseconds SUM=-218 NTIMES=1000000
SIZE=250 TIME=910 milliseconds SUM=-1610 NTIMES=1000000
SIZE=1250 TIME=4328 milliseconds SUM=771 NTIMES=1000000
```

Times multiplying the size by 2:

```
SIZE=10 TIME=155 milliseconds SUM=-199 NTIMES=2000000
SIZE=50 TIME=434 milliseconds SUM=409 NTIMES=2000000
SIZE=250 TIME=1795 milliseconds SUM=1149 NTIMES=2000000
SIZE=1250 TIME=8653 milliseconds SUM=-1519 NTIMES=2000000
```

2- As said before, times are multiplied by k

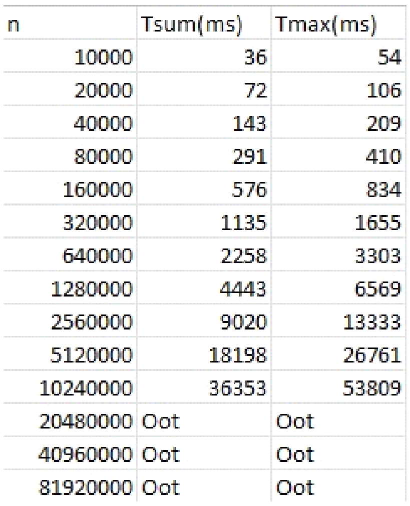
Times multiplying the size by 3:

```
SIZE=10 TIME=238 milliseconds SUM=4 NTIMES=3000000
SIZE=50 TIME=646 milliseconds SUM=-362 NTIMES=3000000
SIZE=250 TIME=2687 milliseconds SUM=-1135 NTIMES=3000000
SIZE=1250 TIME=12978 milliseconds SUM=2597 NTIMES=3000000
```

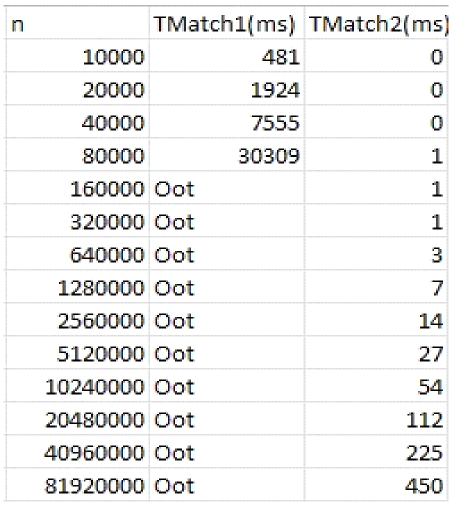
Times multiplying the size by 4:

```
SIZE=10 TIME=312 milliseconds SUM=50 NTIMES=4000000
SIZE=50 TIME=862 milliseconds SUM=-93 NTIMES=4000000
SIZE=250 TIME=3629 milliseconds SUM=830 NTIMES=4000000
SIZE=1250 TIME=17226 milliseconds SUM=-501 NTIMES=4000000
```

3-



n	Tsum(ms)	Tmax(ms)
10000	36	54
20000	72	106
40000	143	209
80000	291	410
160000	576	834
320000	1135	1655
640000	2258	3303
1280000	4443	6569
2560000	9020	13333
5120000	18198	26761
10240000	36353	53809
20480000	Oot	Oot
40960000	Oot	Oot
81920000	Oot	Oot



n	TMatch1(ms)	TMatch2(ms)
10000	481	0
20000	1924	0
40000	7555	0
80000	30309	1
160000	Oot	1
320000	Oot	1
640000	Oot	3
1280000	Oot	7
2560000	Oot	14
5120000	Oot	27
10240000	Oot	54
20480000	Oot	112
40960000	Oot	225
81920000	Oot	450

Computer Intel core i5-12600KF, 32 GB DDR5

It is appreciated that the main difference is in the table of the comparison of the 2 match functions, where match1 has a quadratic complexity, which means that times get really big with a low size. Here we can conclude that linear complexity is much better in terms of performance.

In match2, sum and maximum we can see how the linear complexity works as expected,

increasing as much as the number which the size is multiplied of, for example when the size is multiplied by 2, the times are also multiplied by 2.