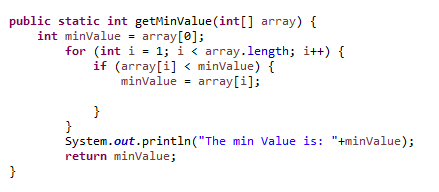
**Report Tutorial course 1 : Algorithmic Complexity**

1. Minimum of an array

To find the minimum element of an array of numerical values, we designed the following algorithm:

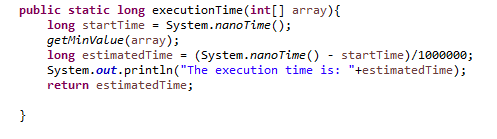


The algorithm will define the first element of the array as the MinValue, thanks to the for loop we run through the array, while browsing the algorithm will compare MinValue with the value of the element in the current position, if it’s under the value of MinValue, it become the new MinValue.

Finding the min value of an array is fast or even instant for moderate array length.

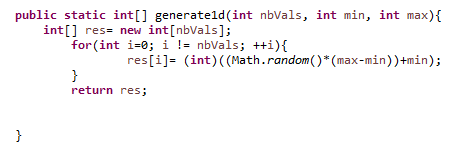
The complexity of this algorithm is linear since we are having only a for loop thus we multiply the running time of statements inside the loop by the total number of iterations.

To calculate the running time, we use the function executionTime:

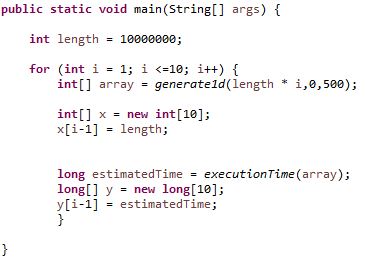


It defines a start time, run the wanted function (here it’s getMinValue), then defines the estimated time where we subtract the start time to get exactly the time of the running algorithm (we divide by 10^6 to get the time in msec and not in nsec).

Using the function generate1d:

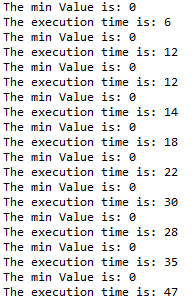


And executing the following code:



We create 10 arrays of length 10^7\*i in order to evaluate to running time of this algorithm.

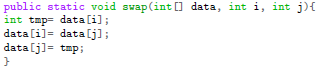
We are obtaining the following results, which lead to the following plot:



As expected, the experimental results match the theoretical complexity of our algorithm since the plot is showing a curve having a linear shape. With a great number of values we might be able to get a perfect line on our graph.

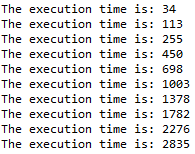
1. Sorting algorithms

The algorithm complexity of the swap function is constant.



Selection Sort:

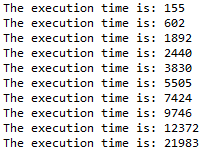
Since we are using two for loops, in minimumIndex then in sort which use minimumIndex function, the evolution of the running time will be quadratic, thus time complexity of our selection sort algorithm will follow a x² curve function depending on the number of elements in our arrays. With the same code than the previous one, we obtain (the full code can be found in the annex):



As expected, we can observe a quadratic evolution of the time needed for the algorithm to sort array with a growing number of elements. (We used array of 10 000 number of elements because with 10^7 elements the algorithm takes too much time to run).

Bubble Sort:

Running the given bubble sort algorithm with our previous code gives the following results:



Since we are having 2 loops again, we expected a quadratic complexity, however we are having major difference between the Selection Sort algorithm and the Bubble Sort algorithm, the last one taking much more time.

It can be explained by the fact that the Bubble Sort algorithm requires on average n/4 swaps per entry with each swap involving two entries (Bubble Sort swap elements of the array 2 by 2). Meanwhile, Selection Sort requires only 1 swap since it run through the whole array looking for the minimum then swap it. The complexity of both algorithms is the same however the Bubble Sort algorithm contains more factors which aren’t included in the global complexity since they are irrelevant to the n² factor.