Algorithms and Data Structures for Biology
25 March 2019 — Lab Session

Ugo Dal Lago Thomas Leventis

1 An Efficient Algorithm to Check Lists

We are interested in the following problem: given a list L of integers and an integer x, we want to check if x is in L. The Python expression `x in L` directly computes the expected result, but do you know how it works? How many times does Python need to read from L? And in general do you think there exists a better solution?

In this exercise we assume the list L to be sorted: for every (valid) indices i and j, if $i \leq j$ then $L[i] \leq L[j]$. We want to find a solution which is *logarithmic* in the size $|L|$ of L, i.e. which will always perform at most $a \log(|L|) + b$ operations for some constants a and b. In other words, we want to show that the algorithm's complexity is $O(\log(|L|)$.

You need to:

- find an adequate algorithm;
- implement this algorithm in Python;
- prove that your code is correct, i.e. it always computes the expected result;
- prove that is has the expected complexity;
- and finally use `cProfile` to experimentally test the complexity of your program; do that by randomly generating the list L and the integer x.

2 Repeated Computations

We now considered a generalisation of the previous problem: given a list L of integers (which we do *not* assume to be sorted) and another list X of integers, we want to know how many indices i there are such that $X[i]$ is in L.

Example. *For* `L = [12,5,46,3,7,11,5]` *and* `X = [1,3,3,46,9,12]`*, exactly* 4 *elements of* X *are in* L*.*

Find algorithms to efficiently solve this problem. How should one proceed if $|L|$ is much larger than $|X|$? If $|X|$ is much larger than $|L|$? If both lists are of similar sizes?

For each of your algorithms:

- implement it in Python;
- prove its correctness;
- describe its complexity, parameterised by $|L|$ and $|X|$;
- test this complexity using `cProfile`. Again, do that by randomly generating the lists L and X.