Programming and algorithms

Practice 4

Objectives for the practice:

- Implement sorting algorithms
- See what complexity is in practice
- Search in a sorted array

As always, encode all your functions in a file.

1 Sorting algorithm

Task 1 *Implement the insertion or bubble sort (pseudo-code given in App-A) that you have seen in class. Don't forget to check that your code is correct by testing it against a few well-chosen examples.*

Task 2 *Implement the heapsort seen during the tutorial. Again, don't forget to check the correctness using examples.*

2 Complexity in practice

Question 1 *Which of the two sorting algorithm you encoded is more efficient ? Can you see why ?*

Task 3 *Using the* `time` *package, get the execution time for each of those algorithms. To do so, simply get the time before the execution and substract it to the time after the execution. The function* `time()` *in the package* `time` *shall be used. Is the result coherent with what you answered in question-1 ?*

To check that one algorithm is faster than another, it is not enough to check on a single example.

Task 4 *Using the* `time` *package, check which algorithm is fastest by testing them on random inputs of size* $n = 2^l$ *for* l *between* 3 *and* 12.
Help: the numpy instruction `np.random.randint(k,size=n)` *will generate a random array of size* n *containing integers in* $[0, k-1]$.

Task 5 *Based on the previous task, output a graphic showing the execution time for each algorithm depending on the input size. Appendix-B explains how to output such a graphic.*

3 Search in sorted array

Task 6 *Implement a sequential search.*

Task 7 *Implement a dichotomous search (only works for sorted arrays).*

Question 2 *In practice, which search algorithm is more efficient on sorted arrays ? Get a graphic to illustrate this as in task-5.*

Task 8 *(optional) If you execute multiple times the previous code, you may have seen that it is not very stable – i.e. the graphs are not exactly the same at each execution. In order to avoid that, realize each execution multiple times and do an average over thoses execution times to have more stable data.*

4 (optional) Using the default python sort

Task 9 *Find how to sort an array with the sort method in python.*

Task 10 *Check the execution time of this sorting algorithm agains the heapsort.*

The default sorting algorithm should be more efficient because it uses a different algorithm (called timsort) which has been very well optimized for most of real-world cases.

5 (optional) First approach to sequence alignment

The problem of sequence alignement is coming from biology and will be our focus for the end of this practice as well as next week's practice. Using wikipedia:

> "In bioinformatics, a sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences."

Task 11 *Look online what sequence alignment is and think about a way to program it. The solution doesn't need to be optimal, it simply needs to work. Call the teacher once you have an idea.*

Task 12 *Code your program, check that it is correct using known examples.*

A Sorting algorithms

function INSERTION_SORT($arr[1..n]$: array_t) : array_t
    $i \leftarrow 1$
    while $i < n$ **do**
        $x \leftarrow arr[i]$
        $j \leftarrow i$
        while $j > 0$ **and** $arr[j-1] > x$ **do**
            $arr[j] \leftarrow arr[j-1]$
            $j \leftarrow j - 1$
        end while
        $arr[j] \leftarrow x$
        $i \leftarrow i + 1$
    end while
    return $arr[1..n]$
end function

function BUBBLE_SORT($arr[1..n]$: array_t) : array_t
    $i \leftarrow 1$
    while $i \leq n$ **do**
        $j \leftarrow i$
        while $j \geq 0$ **and** $arr[j-1] > arr[j]$ **do**
            $arr[j], arr[j-1] \leftarrow arr[j-1], arr[j]$
            $j \leftarrow j - 1$
        end while
        $i \leftarrow i + 1$
    end while
    return $arr[1..n]$
end function

B Graphic

Given x an array of inputs, and fx and gx two arrays of outputs, the graph showing fx and gx as functions of x can be printed as follows (with blue color for fx and red color for gx):

```
import matplotlib.pyplot as plt
plt.plot(x, fx, "blue")
plt.plot(x, gx, "red")
plt.show()
```

You can look online if you want to add legends, axis names ...