Algorithms & Data Structures
SI3 - Polytech Nice-Sophia - Edition 2018

Lab #6: Sorting

This assignment will give you practice about sorting methods. For all parts you are to use the provided classes:

- lab6.py: a testing class. This class provides some interactive code you can use to build lists of `SlowInteger` and sort them
- slowinteger.py: a class for slow comparison integer

Part 1: simple sorting methods

In this part you have to implement the two most common quadratic sorting algorithms:

- *selection sort*
- *insertion sort*

Complete the function templates `selectionSort` and `insertionSort` . For both functions, give the best and worse cases running time complexity and give a list configuration when they occur.

Part 2: merge sort method

In this part you have to implement the recursive *merge sort* algorithm. This algorithm is not in place and needs to use a second list of same size as the list to sort. Complete the function template `mergeSort` .

Part 3: quick sort method

In this part you have to implement the recursive *quick sort* algorithm. This algorithm is based on a partition method which must be carefully designed. Your implementation should follow these requirements:

- the partition method must be the one describes below
- you algorithm should switch to insertion sort algorithm from part 1 below a *cutoff* you have to find yourself by experimenting (default value is 10)
- you have to provided a new insertion sort method to match the new context (i.e. this new method should have more than one parameter)

The partition method should work as follow:

- choose the pivot by picking up a random value in the list slice
- swap the pivot with the first item in the list slice

- partition using a loop which matches the following invariant:

For all `a` in `[i+1, p]`, `b` in `[p+1, k[`, `T[a] < T[i] <= T[b]`

Part 4: heap sort method

In this part you have to implement the *heap sort* algorithm using the `BinaryHeap` class you designed in lab #5. First, you try to figure out how to design a simple but effective sorting algorithm using a binary heap. Then, provide a new version of the class BinaryHeap to match the requirements of your sorting algorithm. Finally, complete function the template `heapSort`.