Problem Set 5 - Searching and sorting

Hello students,
during last lecture you learned about different searching and sorting algorithms. Today, we will deepen your understanding by doing some analysis on them.

For each assignment provide a file showcasing the result of your work: it can be a text file if it is a question or a "*main*" file for the code, which showcases a call to each modified function or method.

Assignment 1 - Algorithm analysis

1. Given an unsorted list of **n-values**, what is the time-complexity to find the k^{th} smallest value in the worst case? What would be the complexity if the list were sorted?
2. What is the O(·) for the **findSortedPosition**() function in the worst case?
3. Consider the new implementation of the **Set class** using a sorted list with the binary search.
    a. Prove or show the worst case time for the **add**() method is O(n).
    b. What is the best case time for the **add**() method?
4. Determine the worst case time complexity for each method of the **Map ADT** implemented in Section 3.2.
5. Evaluate the **insertion sort algorithm** to determine the best case and the worst case time complexities.

Assignment 2 - Modify the algorithms

1. Modify the **binary search** algorithm to find the position of the first occurrence of a value that can occur multiple times in the ordered list. Verify your algorithm is still O(log n).
2. Design and implement a function to find all negative values within a given list. Your function should return a new list containing the negative values. When does the worst case occur and what is the run time for that case?
3. In this chapter, we used a modified version of the **mergeSortedLists**() function to develop a linear time **union**() operation for our **Set ADT** implemented using a **sorted list**. Use a similar approach to implement new linear time versions of the **isSubsetOf**(), **intersect**(), and **difference**() methods.

Assignment 3 - Manually sort

1. Given the following list of keys (80, 7, 24, 16, 43, 91, 35, 2, 19, 72), show the contents of the array after each iteration of the outer loop for the indicated algorithm when sorting in ascending order.
    a. **bubble sor**t
    b. **selection sort**
    c. **insertion sort**
2. Given the following list of keys (3, 18, 29, 32, 39, 44, 67, 75), show the contents of the array after each iteration of the outer loop for the
    a. **bubble sort**
    b. **selection sort**
    c. **insertion sort**

Assignment 4 - Sort classes

When learning sorting algorithms, almost every example uses integers. However, when programming this is rarely the case. Usually, you will work with instances of a class. In Python it is possible to sort classes by providing a key and using the *“sorted”* built-in method. However, not all language offer this versatility, many of them require the implementation of compare function. In this exercise you are asked to implement one of such methods to understand the concept.

Create a file which will allow to test the proposed program.

Transcribe the **insertion sort** example (listing 5.7, page 140 of the book).

Create a class representing a person, with:

1) **first name**
2) **last name**
3) **age** (as an integer)

Language are developed to be useful, and they always allow to sort collection of custom objects. This is done by defining a **compare function**, which given two objects returns:

- **1** - if the first element is bigger than the second;
- **0** - if they are equal;
- **-1** - if the second is bigger.

Define the following compare functions:

- compare_by_age(self, other) - compares the current object with the “other”
- compare_by_name(self, other) - compares the two strings

Hint: strings can be compared using normal values comparison (>, <) in Python, just remember than “b” is greater than “a”.

Finally modify the **insertion sort** so that it accepts a second parameter, which is the **compare function**.

Create a list of Person and sort them using the modified insertion sort, once by name and once by age.

Verify that the result is the same as using the sorted methods from the Python language.