ID1020 – Lab 3
Sorting

1 Organisation

This lab consists of

a) a preparation test in the form of multiple choice questions (marked with MCQ) and

b) a programming part that is to be *presented orally* in the lab session (redovisning) (marked with a *C*).

All the questions must be answered in Canvas. For each programming task, you need to upload your java files into its designated answer box in Canvas.

1.1 Goals

This lab has the following goals:

- Reason about sorting and inversions
- Consider challenges of non-random access data structures in the context of sorting
- Work with pseudocode and transform it into a real implementation

1.2 Definitions

Array Inversion Let a be a list data structure of length $n > 0$ and let $i, j \in \{0, \ldots, n-1\}$ with $i < j$. We call the pair (i, j) an *inversion* of a's if and only if $a[i] > a[j]$.

Example Consider

$$\begin{aligned} \text{indexes: } & (0,1,2,3,4,5) \\ a = & (1,2,4,3,5,0) \end{aligned}$$

then a has the following inversions:

$$\begin{aligned} \text{set of inversions: } & \{(0,5),(1,5),(2,5),(2,3),(3,5),(4,5)\} \\ \text{values of the inversions: } & \{(1,0),(2,0),(4,0),(4,3),(3,0),(5,0)\} \end{aligned}$$

Notice that the inversions are the pairs of indices, *not* pairs of values. The reason is of course that indices are unique, while value are not necessarily unique.

Algorithm 1 BubbleSort

Require: a is a list data structure of length $n > 0$.

```
R ← n − 2
swapped ← true
while R ≥ 0 and swapped = true do
    swapped ← false
    for i ← 0 to R do
        if a[i] > a[i + 1] then
            swapped ← true
            SWAP(a, i, i + 1)
        end if
    end for
    R ← R − 1
end while
```

Ensure: a is an ordered permutation of $\bar{a}$.

Stable Sorting Let $\bar{a}$ be a list data structure of length $n > 0$.Furthermore let S be a sorting algorithm and a be the ordered permutation of $\bar{a}$ after applying S according to the total order relation $\dot{\leq}$ with associated equality relation $\dot{=}$.
We call S *stable* if and only if for all $x, y \in \bar{a}$ with $\bar{a}[i] = x \dot{=} y = \bar{a}[j]$ and $i < j$ it also holds that for $x = a[k], y = a[l]$ $k < l$.

Example Let our values be from the set $\mathbb{N}_{\mathbb{N}} = \{1_1, 1_2, \ldots, 2_1, 2_2, \ldots, \ldots\}$ (this is equivalent to $\mathbb{N} \times \mathbb{N}$). Let $x_i \dot{\leq} y_j$ for $x_i, y_j \in \mathbb{N}_{\mathbb{N}}$ be defined as $x \leq y$, i.e. we ignore the index when sorting. Consider the following list $\bar{a}$ and two ordered permutations a_S, a_T the result of sorting algorithms S and T respectively:

$$\bar{a} = (3_1, 2_1, 2_2, 4_1, 1_1)$$
$$a_S = (1_1, 2_1, 2_2, 3_1, 4_1)$$
$$a_T = (1_1, 2_2, 2_1, 3_1, 4_1)$$

We call S stable and T not stable.

2 Tasks

2.1 Preparation Test (MCQ) – 40P

Look at the lab assignment in Canvas.

2.2 BubbleSort Implementation (C) – 30P

Implement the BubbleSort that is presented in algorithm 1. In your implementation use a singly linked list data structure and make sure that your data structure doesn't make your BubbleSort implementation less efficient than standard BubbleSort.

Hint: you can use any implementation of linked list but you cannot use those operations that are accessing nodes in two-ways (your usage must comply with singly linked list). The second requirement is total time complexity, don't simply assume all linked list operations are of O(1) complexity. With these requirements if you think java linked list is suitable go ahead and use it otherwise implement linked list yourself or use another implementation.

2.3 Inversion Count (C) – 30P

Implement an algorithm that counts the inversions of your linked list structure (see section 1.2). For full points the algorithm should have linearithmic worst-case time complexity (else 10P reduction). Extend your BubbleSort implementation to count the number of calls to *Swap* and compare with the result of your inversion count. Make sure the result matches with the information from section 1.2.