

# Assignment No. 4: Merge k Ordered Lists Efficiently

**Allocated time:** 2 hours

## Implementation

You are required to implement **correctly** and **efficiently** an  $O(n \cdot \log k)$  method for **merging k sorted sequences**, where  $n$  is the total number of elements. (Hint: use a heap, see seminar no. 2 notes).

Implementation requirements:

- Use linked lists to represent the  $k$  sorted sequences and the output sequence

Input:  $k$  lists of numbers  $\langle a_1^i, a_2^i, \dots, a_{m_i}^i \rangle$ ,  $\sum_{i=1}^k m_i = n$

Output: a permutation of the union of the input sequences:  $a'_1 \leq a'_2 \leq \dots \leq a'_n$

## Minimal requirements for grading

- Prepare a demo for each algorithm implemented.
- Interpret the charts and write your observations in the header (block comments) section at the beginning of your *main.cpp* file.
- We do not accept assignments without code indentation and with code not organized in functions (for example where the entire code is in the main function).
- *The points from the requirements correspond to a correct and complete solution, quality of interpretation from the block comment and the correct answer to the questions from the teacher.*

## Requirements

1. **Generate  $k$  randomly sized sorted lists (having  $n$  elements in total,  $n$  and  $k$  given as parameters) and the merging of 2 lists (5p)**

You will have to show your algorithm (*generation and merging*) works on a small-sized input (e.g.  $k=4$ ,  $n=20$ ).

2. **Adapt *min-heap* operations to work on the new structure (3p)**

You will have to show your algorithm (*merging*) works on a small-sized input (e.g.  $k=4$ ,  $n=20$ ).

3. **Evaluation of the algorithm in average case (2p)**

! Before you start to work on the algorithm's evaluation code, make sure you have a correct algorithm!

We will make the **average case** analysis of the algorithm. Remember that, in the average case, you have to repeat the measurements several times. Since both **k** and **n** may vary, we will make each analysis in turn:

- Choose, in turn, 3 constant values for  $k$  ( **$k_1=5$ ,  $k_2=10$ ,  $k_3=100$** ); generate  $k$  **random** sorted lists for each value of  $k$  so that the number of elements in all the lists  $n$  varies between **100 and 10000**, with a maximum increment of 400 (we suggest 100); run the algorithm for all values of  $n$  (for each value of  $k$ ); generate a chart that represents the **sum of assignments and comparisons** done by the merging algorithm for each value of  $k$  as a curve (total 3 curves).
- Set  $n = 10.000$ ; the value of  $k$  must vary between 10 and 500 with an increment of 10; generate  $k$  **random** sorted lists for each value of  $k$  so that the number of elements in all the lists is 10000; test the merging algorithm for each value of  $k$  and generate a chart that represents the **sum of assignments and comparisons** as a curve.