

## Programming Assignment 1

Handed Out: Feb 3

Due: Feb 16 (Monday, 5 PM)

## 1 Overview

Your programming assignment is to design an efficient algorithm, and implement it in C++, for counting the **number of swaps** in a list of numbers. The problem arises in analyzing relative *rankings*. As motivation, consider a movie rental service such as Netflix. The Netflix recommendation algorithm (based on your past history and using machine learning techniques) computes a *ranking* of the movies to recommend to you. We, or Netflix, would also like to know how good its recommendation system is. How can we quantify the *goodness* of their recommendation algorithm?

Consider an example of five movies labeled  $a, b, c, d, e$ . Suppose also that your true preference order for these movies is  $(a, b, c, d, e)$ ; that is,  $a$  is ranked 1 (most preferred),  $b$  is ranked 2, and  $e$  is ranked 5 (least preferred). Now suppose the ranking computed by Netflix is  $(b, d, a, c, e)$ . Can we put a numerical value on how close is the Netflix ranking to your true ranking?

A commonly used approach to compare two rankings is to count the number of pairs that are **out of order**. In other words, we want to count the number of pairs  $(i, j)$  where  $i$  comes before  $j$  in your ranking, but  $i$  comes *after*  $j$  in Netflix ranking. This is called the **number of swaps**. We can abstract the problem as follows. We assume that there are  $n$  movies, and they have been labeled in the order of your true preferences. That is, your ranking is  $(1, 2, \dots, n)$ . Now, given any other permutation (ranking)  $(a_1, a_2, \dots, a_n)$  where each  $a_i \in \{1, 2, \dots, n\}$ , we say that two indices  $i < j$  **form a swap** if  $a_i > a_j$ . That is, the elements  $a_i$  and  $a_j$  are out of order. Your task is to design and implement an  $O(n \log n)$  time algorithm (worst-case) to count the number of swaps in any permutation  $(a_1, a_2, \dots, a_n)$ . For instance, in the example shown above, the number of swaps is 3: the swapped pairs are  $(2, 1), (4, 1), (4, 3)$ .

Your assignment can be broken down into the following pieces.

1. Implement the naive  $O(n^2)$  algorithm to count the swaps. This algorithm goes over all pairs  $i, j$ , with  $i < j$ , and checks whether  $a_i, a_j$  form an swap pair or not, and simply sums the number of swaps.
2. Design a sub-quadratic algorithm to count the number of swaps. As a first step, you may consider designing an algorithm with worst-case running time of  $O(n(\log n)^2)$ . But your final algorithm should have worst-case time  $O(n \log n)$ .
3. Implement the  $O(n \log n)$  time algorithm.
4. **Both of your algorithms should accept stdin input as a list of space-separated integers, and output the swap count.** This is how the TA will test the correctness of your algorithm, by feeding it some permutations and looking at the output count.

5. Finally, you should perform a comparative empirical analysis to see how much faster the  $O(n \log n)$  time algorithm runs in practice compared to the  $O(n^2)$  time algorithm. To do this, generate 100 random permutations for each of the following values of  $n$ :  $n = 1000$ ,  $n = 10,000$ , and  $n = 100,000$ . For each value of  $n$ , plot the **average** running time over the 100 runs, for both the  $O(n \log n)$  algorithm as well as the  $O(n^2)$  time algorithm. How closely do the theoretical worst-case bound agree with your observed times? Comment on your experimental results.

## 2 Readme

Provide a Readme file, explaining how to compile, and use your program. Without this file your project will not be graded and you will get no points. In this file you should have your *name*, *email address* and you should say which parts of your project work and which ones do not. The project should be implemented as it is described here, so if the input or output format for your program does not adhere to the guidelines described, you will not get any credit for that part of your program.