Exercise 2 Sorting

Algorithm

Assume that you have *n* integer numbers that >= 1 and <= *k*. You need to sort them in O(*n*) complexity.

Hint – use an array of size *k*, initialize the array elements to zero.

Let's name array as **A**. Use numbers from the list as array indexes. When you read a number, simply add 1 to the array element at the appropriate index.

Pseudocode for the algorithm will be

For **number** in list

    A[number] = **A[number]** + 1;

End For

By the end of this loop, the list will be sorted, and all repetitions will be also counted. Now you just need to have another loop to print results from **A**.

For **index** in range 1**..***k*

    If **A[index]** is not zero, then output the **index** as many times as the value of **A[index]**

End For

The time complexity of this algorithm can be estimated as follows

- Array A initialization is O(*k*)
- Storing input numbers in array A is O(*n*)
- Printing results – checking all array elements is O(k), non-zero elements output is O(*n*).

So, the total complexity is O(*k*+*n*). Assuming that *k* is a constant, the complexity is O(*n*).

What if *k* is not a constant? In such a case, we will have to use some kind of dynamic storage instead of the array, for example, a vector. The vector is usually implemented as an array that has some spare capacity. When the array is filled, a new array of larger size is created, and all data of the old array is copied to the new array. The new array size is usually two times the old array size.

The vector will grow in size as needed, but it means that the vector's elements will be periodically copied that is time consuming. Can you estimate the time complexity as function of *k* and *n*?

Exercise2 Program

Assume that you have *n* arbitrary numbers in range 1..*k*. The integers may be repeated. Sort them using the algorithm of O(*n*) complexity. Also sort them using any other algorithm of your choice.

Write a program does both of the following

- reads numbers from input file **in*X*.txt**, sorts them using O(*n*) algorithm, and output the sorted numbers to output file **out*X*a.txt** (where *X* is 1, 2, …)
- reads numbers again from the same input file, sorts them using any other algorithm of your choice, and output the sorted numbers to output file **out*X*b.txt**

There are two test input files provided – **in10.txt** and **in100.txt**. There is also one sample output file corresponding to the first input file – **out10_sample.txt** (file **out100_sample.txt** is not provided).

Run your program with the test input files. For each input file, two corresponding output files (produced by your program) have to be the same.

Also compare the produced output file **out10a.txt** with the sample file **out10_sample.txt**. Submit the test input and the produced output files together with the program source code.

You will have a lot of holes in your array, but it is OK, you waste in memory but win in time.

Always win? When is better to use, for example, merge sort?

Input File Format

The input file contains non-negative integer numbers separated by whitespaces

- the first number is *k (maximum possible number below)*
- the follow up numbers are the numbers that you need to sort (count these numbers to figure out *n*)

For example,

100 // so *k* is 100
5 33 10 17 // so *n* is 4

In this example, the actual maximum number is 33 that can be calculated after all numbers are read. But we have to allocate array at the beginning, so we specify k == 100.

