Project 2:

DISTRIBUTION SORTING ALGORITHM. 2ND PART

OBJECTIVES

- Understand how the distribution sorting algorithms work.
- Optimize the sorting algorithms.
- Comparative study of temporal costs of the proposed algorithms.

TO BE HANDED OVER

- sort_test2.cpp, sort_stl.cpp
- makefile (without parameters should compile all programs)
- Optimizations done in section 1 and measured time of each optimization.
- Comparison of measured time for each algorithm and conclusions.

DEVELOPMENT

In this project we will make a comparative study of the sorting algorithms implemented in the previous project and some sorting algorithms of the STL. Before that, we will have to optimize the algorithms of the previous project.

In case you know or want to learn how to use *git*, it is a good idea to use it for this project. At the appendix, you have a small subset of commands for using *git*.

1. Optimization of the algorithms

Each algorithm should be optimized separately (countsort, radixsort for vector and radixsort for list). We will generate a vector of 10.000 elements with a range of [0..65535] and sort a **copy** of the vector. We will have to repeat enough times the sorting operation to be able to measure the execution time precisely. It is recommended to use *clock_gettime* or the *chrono* library for measuring the time.

For each optimization the execution time should be tested (compiling with option: –O2). If the optimization does not improve the program (in time or space) it should be discarded.

In a pdf document you will have to explain the optimizations done for each algorithm, the execution time obtained after each optimization and whether the optimization has been maintained in the program or has been discarded.

2. Binary RadixSort

After all the optimizations, there is a last optimization to be done that consists of doing the sorting in base 16, i.e., we will have 16 buckets instead of 10. The advantage of using base 16 is that the "digit" corresponds exactly to 4 bits, so we can obtain the digit doing shifts. This optimization should be applied to **both versions** of radixsort. These new versions of the radixsort will be implemented in 2 new functions.

3. Comparison with sorting algorithms of the STL

Write a new program (*sort_stl.cpp*) that will do the sorting using the sorting algorithms present in the STL. For sorting the vectors it should be used the ***sort*** function of the *algorithm* library. For sorting the lists it should be used the ***sort*** **method**.

Compare the execution times of all the algorithms compiling with the option **–O2**. The execution times should be obtained for the ranges [0..255] and [0..65535] for all the algorithms (countsort, binary radixsort for vectors and for lists, and STL sort algorithms for vectors and for lists). The execution times should also be obtained for vector (or list) sizes **1.000**, **10.000** and **100.000**. Show all the results in a graphic or a table.

What conclusions have you obtained from the comparative?

Appendix 1: GIT very basic commands

Configuring git:

```
$ git config --global user.name "[your name]"
```

Creating local repository:

```
$ git init [project name]
```

Adding the files of your project

```
$ git add [file]
```

After each change to your project, you record the present state:

```
$ git commit –a "[descriptive message]"
```

You can also install and use a graphical user interface for git: *git-gui*