Analisys of Algorithms
Escuela Politécnica Superior, UAM, 2020–2021

Set of practices n. 2

Date of Delivery

- Monday Groups: 15 de Noviembre.
- Wednsday Groups: 17 de Noviembre.
- Friday Groups: 19 de Noviembre.

This second practice tries to experimentally determine the execution times of algorithms which use divide and conquer approach. The algorithsm to be studied are *MergeSort and QuickSort*. Of each algorithm it will be analyzed on tables of different sizes and the results obtained will be compared with the theoretical study of algorithm.

MergeSort Algorithm

1. Implement in the file `sorting.c` the sorting algorithm corresponding to the method known as `MergeSort`. The prototype of the necessary functions will be int `mergesort(int* table, int ip, int iu)` and `int merge(int* table, int ip, int iu, int imiddle)`. Both functions return `ERR` in case of error or the number of times the BO has been executed in case the table is ordered rightly, `table` is the table to sort, `ip` is the first element of the table, `iu` is the last element of the table, `imiddle` is the index of the midpoint of the table.

   Using the file `exercise4.c` of the previous practice, check that *MergeSort* algorithm orders correctly.

2. Adapting the file `exercise5.c` of the previous practice for using *MergeSort* algorithm, obtain the table with the average clock time, and the average, minimum and maximum number of times that the BO is executed depending on the size of the permutation. Plot these values and compare them with the theoretical result of the algorithm.

QuickSort Algorithm

3. Implement int `quicksort(int* table, int ip, int iu)` corresponding to the method known as *QuickSort*; This routine returns `ERR` in case of error or the number of basic operations in case the table is ordered rightly,where `table` is the table to sort, `ip` is the first element of the table and `iu` is the last element of the table.

   The routines `partition` and `median` to support `quicksort`, are necessary to do an efficient sorting, and they must be declared as `int split(int* table, int ip, int iu,int *pos)` and `int median(int *table, int ip, int iu,int *pos)`. Both routines will return `ERR` in case of error or the number of basic operations in case of table is ordered correctly, the position of the pivot will be returned through the argument of type pointer `pos`. In this first implementation, we will use the first element of the table as a pivot, therefore, median must return value `0` (or `ERR` in case of error) and return the `ip` value through the argument `pos`.

   Using the `exercise4.c` of the previous practice, check that *QuickSort* algorithm orders correctly.

4. Adapting the file *exercise5.c* of the previous practice for using *QuickSort* algorithm, obtain the table with the average clock time, and the average, minimum and maximum number of times that the BO is executed depending on the size of the permutation. Plot these values and compare them with the theoretical result of the algorithm.

5. One of the aspects that most infludes in the performance of a routine is the overhead produced by a high number of function calls. Try to check this situation by modifying the `quicksort` routine in such to remove tail recursion. Name the new function without tail recursion *quicksort_ntr*.

   Check the performance of the routine `quicksort_ntr` and compare it with the one of the original routine `quicksort`.

Questions

1. Compare the empirical performance of the algorithms with the theoretical average case for each case. If the traces of the performance graphs are very sharp, why do you think this happens?.
2. Analyze the results obtained when comparing `quicksort` con `quicksort_src` both in the cases you observe differences or not.
3. What are the best and worst cases for each algorithm? What should be modified in practice to strictly calculate each case (include average case too)?
4. Which of the two algorithms studied is empirically more efficient? Compare this result with the theoretical prediction. Which algorithm(s) is/are more efficient from the point of view of memroy management? Justify your answer.

Material to be delivered in each of the sections

Documentation: The documentation will consist of the following sections:

1. **Introduction:** It consists of a technical description of the work to be carried out, what are the objectives they intend to reach, what input data is required and what data is obtained from the output, so like any kind of comment about the practice.
2. **Printed Code:** The routine code according to the corresponding section. The code also includes the header of the routine.
3. **Results:** Description of the results obtained, comparative graphics of the results obtained with the theoretical ones and comments on them.
4. **Questions:** Answers to theoretical questions.

The printed documentation will be given to the practice teacher in the class corresponding to the day after delivery of the practice. On the cover should be included The names of the students. All the files necessary to compile the practice and documentation have to be stored in a single compressed file, in zip format, or tgz (tgz represents a tar file compressed with gzip). The name of that file will be **name1 last_name1 name2 last_name2.zip** or **name1 last_name1 name2 last_name2.tgz**. Where **name last_name** is the name and last name of each of the members of the couple.
Additionally, the practices should be stored in some storage medium. (pendrive, CD or DVD, hard disk, remote virtual disk, etc.) by the student for the day of the practice exam in December.

Warning: The importance of carrying a pendrive is stressed **in addition to other storage media such as usb disks, cd, remote virtual disk, email to own address, etc**, since it is not guaranteed that they can be mounted and accessed each and every one of them during the exam, which will mean the grade not pass s practices.

In order to normalize the compilation and execution of the various programs, it is suggested use the make tool next to the Makefile file included in the file **exercise1.zip**. In this Makefile file the options exercise1, exercise2, ..., exercise5 will be implemented that compile the programs of the different sections. The exercise1_test, exercise2_test, ..., exercise5_test options will also be implemented to run the programs. You can also use development environments such as Netbeans, Anjuta, Visual Studio, etc.

Instructions for the delivery of the practice codes

The delivery of the source codes corresponding to the practices of the subject AA will be carried out through the Web page **https://moodle.uam.es/**.