PLATYPUS

ESILV LÉONARD DE VINCI | ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

ESILV 4A ADSA
Practical Work 6
Complexity & divide and conquer paradigm

>> Instructions

It is important to analyse the posed problems and conceive your algorithms before having them validated by your teachers.

Just go all-out in the code is always a bad strategy.

Work to deliver by email at the address of your teacher, at the latest at 23:59, the 4th day following the session.

You must deliver an archive named:

The object of your email and the name of your deliverable are:

ESILV TP 6 BT ADSA 4A [GROUP] - [NAMELIST OF THE MEMBERS OF THE TEAM]

Your deliverable presents itself in the form of an archive containing all the C++ source files (no executable) completed by a README.txt file containing a report on progress of your work and the main difficulties you encountered in around 10 lines.

These instructions are completed by the Computer Work Delivery Instructions available online on the Google Drive sharing.

>> Assessment

The main assessment points are:

- For the executable
  - Functional coverage, according to what has been asked.
  - Readability of the execution traces.
  - Ease of use and ergonomics of the interaction.
  - For the code
  - Quality of the data structures and algorithms.
  - Quality of the code itself (indentations, comments, choice of identifiers, etc.)
  - Thoroughness is also taken into account.
  - Respect of the delivery instructions, especially the deadline.
  - Quality of the README.

Note: a program that does not compile or bugs cannot get a mark above 10/20 (results/driven culture)

The average mark for each group turns around 12.
There is no standard-deviation constraint and the best works can get 20.

Conceive and realise in C++ :

>> Part to be realized during the session

1) A recursive algorithm (resp. iterative) that takes as argument an integer *n* and compute its factorial. Estimate its complexity.
2) A recursive algorithm (resp. iterative) that takes as arguments two integers, *a* and *n*, and compute a^n (*a* power *n*). Estimate its complexity.
3) A recursive algorithm (resp. iterative) that takes as arguments two integers, *a* and *n*, and compute their PGCD (Euclid algorithm). Estimate its complexity.
4) Main element:
   Given a x element in a T array of size n (if contains more than n / 2 occurrences of x), using the only operation of comparison:
   1) Conceive and realize an algorithm NrOccs, that takes as arguments a value x and two indexes I and j, and counts the number of occurrences of x between *T[i]* and *T[j]*. Estimate its complexity in terms of number of comparisons.
   2) Conceive and realize an algorithm that checks if a *T* array contains a main element and then returns it, otherwise returns NIL. Estimate its complexity.
5) An algorithm *min_TAB*, which, from a *TAB* unordered array of *n* integers, returns the lesser element of *TAB*. Estimate its complexity.
6) An algorithm of insertion sort (iterative) that sorts in growing order, a *TAB* array of n integers. Estimate its complexity.

>> Part to be realized at home

7) Update all recursive algorithms of previous section (question 1, 2, 3, 4) using the divide & conquer paradigm.
8) An algorithm that searches for the index, in an ordered *TAB* array of integers, pairwise distinct (in growing order), of a given element (returns 0 if *TAB* doesn't contain this element).
9) A MERGE_SORT algorithm that takes into parameter a *TAB* array of size *N* and two integers *a* and *b* (exp. *a* = 0 and *b* = *N* – 1) and sorts that array using the divide and conquer paradigm.
10) Produce a performance benchmark report (comparing naïve algorithms of first section and those using the divide and conquer paradigm).