Ministerul Educaţiei și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work 2:
Study and Empirical Analysis of Algorithms for Studying divide et impera

Elaborated:
st. gr. FAF-211 Echim Mihail

Verified:
asist. univ. Fiștic Cristofor

Chişinău - 2023

ALGORITHM ANALYSIS

Objective
Study and analyze different divide et impera sorting algorithms.

Tasks:

1. Implement 3 divide et impera sorting algorithms and 1 by choice;
2. Decide properties of input format that will be used for algorithm analysis;
3. Decide the comparison metric for the algorithms;
4. Analyze empirically the algorithms;
5. Present the results of the obtained data;
6. Deduce conclusions of the laboratory.

Theoretical Notes:

An alternative to the mathematical analysis of complexity is empirical analysis.

This may be useful for obtaining preliminary information on the complexity class of an algorithm; comparing the efficiency of two (or more) algorithms for solving the same problems; comparing the efficiency of several implementations of the same algorithm; obtaining information on the efficiency of implementing an algorithm on a particular computer.

In the empirical analysis of an algorithm, the following steps are usually followed:

1. The purpose of the analysis is established.
2. Choose the efficiency metric to be used (number of executions of an operation (s) or time execution of all or part of the algorithm.
3. The properties of the input data in relation to which the analysis is performed are established (data size or specific properties).
4. The algorithm is implemented in a programming language.
5. Generating multiple sets of input data.
6. Run the program for each input data set.
7. The obtained data are analyzed.

The choice of the efficiency measure depends on the purpose of the analysis. If, for example, the aim is to obtain information on the complexity class or even checking the accuracy of a theoretical estimate then it is appropriate to use the number of operations performed. But if the goal is to assess the behavior of the implementation of an algorithm then execution time is appropriate.

After the execution of the program with the test data, the results are recorded and, for the purpose of the analysis, either synthetic quantities (mean, standard deviation, etc.) are calculated or a graph with appropriate pairs of points (i.e. problem size, efficiency measure) is plotted.

Introduction:

The divide-and-conquer paradigm is often used to find an optimal solution of a problem. Its basic idea is to decompose a given problem into two or more similar, but simpler, subproblems, to solve them in turn, and to compose their solutions to solve the given problem. Problems of sufficient simplicity are solved directly. For example, to sort a given list of *n* natural numbers, split it into two lists of about *n*/2 numbers each, sort each of them in turn, and interleave both results appropriately to obtain the sorted version of the given list (see the picture). This approach is known as the merge sort algorithm.

The name "divide and conquer" is sometimes applied to algorithms that reduce each problem to only one sub-problem, such as the binary search algorithm for finding a record in a sorted list (or its analog in numerical computing, the bisection algorithm for root finding). These algorithms can be implemented more efficiently than general divide-and-conquer algorithms; in particular, if they use tail recursion, they can be converted into simple loops. Under this broad definition, however, every algorithm that uses recursion or loops could be regarded as a "divide-and-conquer algorithm". Therefore, some authors consider that the name "divide and conquer" should be used only when each problem may generate two or more subproblems. The name **decrease and conquer** has been proposed instead for the single-subproblem class.

An important application of divide and conquer is in optimization, where if the search space is reduced ("pruned") by a constant factor at each step, the overall algorithm has the same asymptotic complexity as the pruning step, with the constant depending on the pruning factor (by summing the geometric series); this is known as prune and search.

Comparison Metric:

The comparison metric for this laboratory work will be considered the time of execution of each algorithm (T(n))

Input Format:

As input, each algorithm will receive arrays randomly filled with integers bigger than 0 and smaller than 101. The algorithms will be of sizes: 10, 100, 1000, 10000, and 100000. I tried using 10^6 as well, but most algorithms gave either maximum recursion error or memory error, so I decided to stop at 10^5.