TECNOLÓGICO SUPERIOR DE JALISCO
MARIO MOLINA
ZAPOPAN

Data Structures and Algorithms

Ordering

Deadline: 10/20/2017

Practice objective

To code and verify the correct operation and efficiency of some common sorting algorithms.

Background

Assume a set Φ such that for all α and β members of Φ, if $\alpha \neq \beta$ then any $\alpha < \beta$ or $\beta < \alpha$. I.e., *all the elements of Φ are comparable* to each other. Let $\Lambda = a_1 a_2 \cdots a_n$ be a finite sequence of variables with domain in Φ. The *ordering problem* consists in reallocating the values of the variables a_1, a_2, …, a_n in such a way that $a_i \leq a_{i+1}$ holds for all $1 \leq i < n$, making Λ and *ordered list*. If Λ is an ordered list, this is denoted $ord(\Lambda)$.

General guidelines

During the course, some sorting algorithms has been studied: *Bubble Sort*, *Merge Sort* and *Quick Sort*. Assume an n—length array. The order of complexity of such algorithms is as follows:

Algorithm	Average case
Bubble Sort	$O(n^2)$
Merge Sort	$O(n \log_2 n)$
Quick Sort	$O(n \log_2 n)$

Table 1. The temporal complexity of the sorting algorithms.

Your practice will be to implement each of these algorithms and verify both its behaviour (the functional verification) and its temporal complexy (the efficiency verification, **Table 1**), as explained below.

Development of the practice

Functional verification. It is to verify the correctness of the algorithm, i.e., to verify if it really works. That is, given an arbitrary list $\Lambda = a_1 a_2 \cdots a_n$ your algorithm should transform it into $ord(\Lambda)$. To do that, your program should allow the following:

1. Ask the user for te value of n. Your program will generate and display an n—length list initialized with random values.
2. Ask for the sorting algorithm to be used. Your program will sort and then display the list previously generated with the algoritm that was selected.

Every time a new list is generated, the range of the random values must be in the interval from 1 to $10n$. For example, for $n = 100$ the values in the list must be ramdomly selected between 1 and 1000.

Efficiency verification. It is to verify the effort (the number of operations) that an algorithm requires in order to sort a randomly generated n—length list. To compute it, your program should do the following:

1. Ask the user for the values n and m.
2. Ask for the algorithm to be used.
3. Display the results shown below.

Your program should generate and sort an n—list m times. It will count the times the fundamental operation is executed during the ordering and calculate its average[1]. Suppose, for example that quicksort is executed 100 times ($m = 100$) for lists of length 1000 ($n = 1000$). Then, your count shuld be similar to 7106, 7788, 6272, 8070, etc., and your average around 7642. *Do not print* these lists, yo will only need the result.

[1] Note that each time you run a new test for an n—list, you should do so with a new one. Otherwise you will always get the same result and the average of the m ordinances will not be representative.

Practice report

For your report you will use your efficiency verification to fill the next table for each of the algorithms:

Algorithm: _____		
n	Average effort (100 samples)	Efficiency function ($n \log_2 n$)
1000	7554	9965
2000	17060	21931
…	…	…
20000	232941	285754

Plot these results and discuss. How are the average effort and the efficiency function related? How have you done to locate the fundamental operation? Is there a general criterion to do it? What does the efficiency function say about the algorithm? How do you know that one algorithm is better than another only by reading their efficiency function?

Make a conclusion.

Reference values

You can use the following reference values for your own results. In the **Table 2** are shown the values of the efficiency functions.

n	n^2	$n \log_2 n$	n	n^2	$n \log_2 n$
1000	1000000	9966	11000	121000000	147677
2000	4000000	21932	12000	144000000	162609
3000	9000000	34652	13000	169000000	177661
4000	16000000	47863	14000	196000000	192824
5000	25000000	61439	15000	225000000	208090
6000	36000000	75304	16000	256000000	223453
7000	49000000	89412	17000	289000000	238905
8000	64000000	103726	18000	324000000	254443
9000	81000000	118221	19000	361000000	270061
10000	100000000	132877	20000	400000000	285754

Table 2. Reference values for the temporal complexity functions.

Table 3 shows the approximate actual values your program should generate for each algorithm:

Algorithm	Approximate value of the effort with respect to its efficiency function
Bubble Sort	~50%
Quick Sort	~79%
Merge Sort	~100%

Table 3. Expected values

Delivery

The practice reporte must should be printed. Send your code in a .zip, .rar or .7z file named `order.<name>.zip` where *<name>* is *your* name. For example, `order.eddard.stark.zip`. Send your file to aaguilar.itszapopan@gmail.com no later than the deadline with the subject *Order Test*. They will not be reviewed consignments that fail to complay with any of these requirements or if they do not compile. The deadline is on Friday, October 20th.