Advanced algorithmic : Practical session #1

Data Structures and Complexity

1 Supervised Practical Session + 1 Unsupervised Practical Session + 1 Questionnaire

---

The exercises should be progammed in **C language**. The basics of **C** language and compilation/execution are considered as pre-requisite.

⚠ After the supervised practical session, there is an unsupervised practical session, then at the next supervised practical session, a questionnaire is organized to evaluate the achievement of the goals.

Context

We would like to develop an efficient algorithm for extracting the k greatest elements in an array of n integers with $1 \leq k < n$ and returning them in increasing order. By efficient, we mean that the process should be done in-place with a spatial complexity in $O(1)$ and the smallest temporal complexity as possible.

This sequence of practical sessions aims at implementing in **C** language and evaluating the different possible approaches to solve the problem.

The problem data are read from the standard input or from a text file following the following format :

— an integer giving the value of n,
— an integer giving the value of k,
— n integers corresponding to the data.

The program should take an optional argument for the name of the input file (without argument : the data are read from the standard input stream).

After runtime, the program should print the k selected values under the form of an integer list separated with blank spaces. For instance, the following test file :

```
7 3
30 23 12 9 1 2 50
```

contains 7 integers from which the 3 greatest ones should be extracted.

After runtime, the standard output contains the values 50 30 23.

On moodle you can find the following files :

— `test7.txt` contains the previous example with 7 elements,
— `data10000.txt` is a test file with 10 000 integers
— a file `tpSD.c` to finish off, this program reads and print the array taken from a file that is given as input parameter or that is entered on standard input. In order to generate the executable program `tpSD`, enter : `gcc tpSD.c -o tpSD`
In order to run it enter : `tpSD test7.txt`
— an example of use of a timer in C

Exercise 1 : Simple Approach with bubble sort

The bubble sort algorithm applied to an integer array of size n is based on an intern loop that pushes the greatest value at the end of the array by successive comparison of an element with the next one and by swapping them if necessary. The extern loops of this algorithm processes the intern loops for all the size from n downto 1. By doing the extern loop only k times, the k greatest elements will be brought to the end of the array and will be printed by our program.

— Implement this simple solution and validate it on the tests that are given.

Exercise 2 : Use of a maximal binary heap

If the data are structured in a maximal binary heap, the acces to the k greatest values by decreasing order amounts to do k times the REMOVE operation on the heap.

1. Implement the BUILDHEAP and REMOVE functions on a maximal binary heap represented by an array.
2. Write a program that extracts the k greatest values of an array of size n by using your maximal heap.
3. Implement the PERCOLATEUP and ADD functions for a maximal heap. The function ADD inserts an element only if the number of elements of the binary heap (called SIZE) is lower than the capacity of the heap (called LENGTH).
4. Create an empty static array of size LENGTH (for instance, the number n of elements that are in the test file + 5). Fill in the array by building a maximal binary heap with n successive calls to ADD with the elements of the array being the integers read from the file. Then extract the k greatest values of this array of n elements by doing k calls to REMOVE.
5. Compare experimentally the time spent by the three ways to extract the k greatest values : the method of Exercise 1 and the two methods of Exercise 2, for several values of k and n.

Exercise 3 : Full sorting of the set

When the data is sorted, it is very easy to extract the k greatest elements in increasing order.

1. By using a sort algorithm of your choice, implement the solution based on sorting the data then extract the k greatest values. You can use existing sort implementation, for instance, the qsort function of the C standard library.
2. Compare experimentally the efficiency of the two methods : fo a fixed value of n great enough (10^7 for instance), find the value of k from which the bubble partial sort becomes slower than the complete sort. You can use the timer as done in the proposed example on moodle.

Work to do before next supervised session

The questionnaire will ask you to run the programs done for the 3 exercises on small and big files in order to give the k greatest integer of an array.

You should be able to make some small changes on your programs and to answer to some variants of the questions of this document. You should also be able to print the binary heap that is built.

There will be some questions about programs complexity in the questionnaire in particular you should be able to answer to the following questions :

— Give an order of magnitude of the temporal complexity of your algorihm in the worse case for the algorithme of Exercise 1 (bubble sort) : first express the time taken by the algorithm with a function $f(n, k)$ where n is the size n of the array and k is the number of elements, then propose a function $g(n, k)$ such that $f \in \Theta(g)$.

— Express the space complexity of the two programs of Exercise 2 (with the Binary Heap) in terms of Θ.

— By using the complexity results about BuildHeap, show that the temporal complexity (with notation $\mathbf{O}$) of extracting the k greatest elements among n with this approach is always smaller than with the approach of Exercise 1.

— By using the known complexity results on average (see e.g. wikipedia) for the sort algorithm that you have used, give the asymptotic complexity of the method for Exercise 3. What relations between k and n should hold in order to make the approach of Exercise 3 more efficient than the approach of Exercise 2 ?