

# Striving for Efficiency in Algorithms: Sorting

Inna Pivkina<sup>\*</sup>

Sorting is the fundamental algorithmic problem in computer science. It is the first step in solving many other algorithmic problems. Donald Knuth, a world famous computer scientist and author of the book “The Art of Computer Programming, Volume 3: Sorting and Searching”, wrote: “I believe that virtually every important aspect of programming arises somewhere in the context of searching or sorting”.

Quicksort was invented by Tony Hoare in 1960. When Hoare became a programmer with a small computer manufacturer, his first task was to implement a sorting routine based on the best then-known algorithm (Shellsort, which is a  $O(n^2)$  algorithm that is a slight improvement on the InsertSort algorithm). Hoare “greatly enjoyed the challenge of maximizing efficiency” and when he told his boss that he had invented a sorting method that would usually run faster than Shellsort, the boss bet him sixpence that he had not. Hoare won his bet. Quicksort is a comparison sort which is significantly faster in practice than other comparison sorting algorithm. Hoare described the algorithm in his papers in 1962 and 1961 ([1] and [2]). After its invention by Hoare, Quicksort has undergone extensive analysis by Robert Sedgewick in 1975, 1977, 1978 ([3], [4], [5]). Sedgewick in his paper "Implementing Quicksort programs" (1978) presented “a practical study of how to implement the Quicksort sorting algorithm and its best variants on real computers”. The paper contains the original version of Quicksort and presents step-by-step modifications to the algorithm which make its implementation on real computers more efficient.

## Project overview:

The project is based on the paper by R. Sedgewick “Implementing Quicksort programs” ([5]). The paper contains the original version of Quicksort and its modification, which as Sedgewick says combines the most effective improvements to Quicksort. The main idea of the project is to experimentally verify results of Sedgewick by implementing the algorithms and comparing their running times.

The project is divided into several parts. Each part contains a reading assignment and a list of tasks.

---

<sup>\*</sup> Department of Computer Science; New Mexico State University; Las Cruces, NM 88003; [ipivkina@cs.nmsu.edu](mailto:ipivkina@cs.nmsu.edu).

## Project Part 1:

Read the paper until section “Worst Case” on page 850 (read sections Introduction, The Algorithm, Improvements, Removing Recursion, Small Subfiles).

1.1) Program 1 in the paper describes original version of Quicksort. The operators and the constructs which are used in Program 1 are different from those which are used the textbook (Cormen et al.). For instance, operator **:=** and the control structures **loop ... repeat**. Rewrite Program 1 using pseudocode conventions from the textbook.

1.2) Partitioning algorithm which is used in Program 1 is different from the version of PARTITION given in the textbook. Write a pseudocode for procedure Partition1 which performs partitioning using the partitioning algorithm from Program1. Use pseudocode conventions from the textbook.

1.3) Demonstrate the operation of Partition1 from the above question on the array  $A = \langle 11, 12, 4, 8, 15, 9, 7, 1, 6, 16 \rangle$ . Show the values of the array and auxiliary values (values of  $i$  and  $j$ ) after each step. (You may use Figure 1 on page 849 of the paper as an example of showing the values of the array.)

1.4) Demonstrate the operation of Program 1 (quicksorting) on the array  $A$  from the above question. You may use Figure 2 on page 849 of the paper as an example.

1.5) Demonstrate the operation of Partition1 from question 1.2 on the array consisting of equal numbers  $A = \langle 3, 3, 3, 3, 3, 3 \rangle$ . Show the values of the array and auxiliary values (values of  $i$  and  $j$ ) after each step. (You may use Figure 1 on page 849 of the paper as an example of showing the values of the array.) Notice that this example fits the description of the partitioning process from the last paragraph of the first column on page 848 of the paper which starts with words “If equal keys are present among  $A[1], \dots, A[N] \dots$ ”.

1.6) Give an example of array  $A$  such that:

- $A$  has 5 elements, and
- exactly 2 out of these 5 elements are equal, and
- using Partition1 on  $A$  fits the description of the partitioning process from the last paragraph of the first column on page 848 of the paper which starts with words “If equal keys are present among  $A[1], \dots, A[N] \dots$ ”.

Demonstrate the operation Partition 1 on this array  $A$ .

1.7) On page 848 Sedgewick writes: “For example, rather than using the “sentinel”  $A[N+1] = \infty$  we could use

**loop:**  $i := i + 1$ ; **while**  $i \leq N$  **and**  $A[i] < v$  **repeat;**

for the  $i$  pointer increment, but this would be far less efficient.”

Explain why this would be far less efficient.

1.8) What strategy does Sedgewick adopt on the question of how keys equal to the partitioning element should be treated?

1.9) On page 849 Sedgewick says: “For example, if the file  $A[1], \dots, A[N]$  is already in order, then the program will invoke itself to recursive depth  $N$ ”.  
Verify the statement by drawing tree of recursive calls for execution of Program 1 on already sorted array  $A[1], \dots, A[N]$  (assume that all the elements being sorted are distinct).

1.10) Rewrite procedure *insertionsort* on pages 849-850 of the paper using pseudocode conventions from the textbook. Notice that it is different from the insertion sort implementation in the textbook.

1.11) Demonstrate the procedure *insertionsort* from the above question on the array  $A = \langle 11, 12, 4, 8, 15, 9, 7, 1, 6, 16 \rangle$ . Show the contents of the array after each execution of the outer loop.

1.12) On page 850 Sedgewick writes: “The obvious way to improve Program 1 is to change the first if statement to

**if**  $r-l \leq M$  **then** *insertionsort*( $l, r$ ) **else** ...”.

Why this is an improvement?

1.13) What is an even better way to modify Program 1 than the one in question 1.12?

1.14) What is the best value of the length of unsorted subfiles?

### Project Part 2:

Read the paper until section “Assembly Language” on page 852 (read sections Worst Case, Median-of-three Modification, Implementation).

2.1) What method did Hoare suggest to make the worst case unlikely to occur in practice? Explain in your own words why this method works.

2.2) What modification to Program 1 did Sedgewick suggest to implement the method from 2.1? When describing the modification please use pseudocode conventions from the textbook.

2.2) Describe in your own words the idea of median-of-three modification. What is the purpose of this modification?

2.3) What implementation of median-of-three modification does Sedgewick present in the paper? When describing the implementation please use pseudocode conventions from the textbook.

2.4) Rewrite Program 2 using pseudocode conventions from the textbook.

2.5) Explain why condition  $A[N+1] = \infty$  is needed in Program 2? What will happen if the condition is not there?

2.6) Demonstrate the operation of Program 2 upon the digits of constant e, that is,  $A = \langle 2, 7, 1, 8, 2, 8, 1, 8, 2, 8, 4, 5, 9, 0, 4, 5 \rangle$ . Use  $M=4$ . You may use Figure 6 on page 852 of the paper as an example. (This question is closely related to the next question 2.7.)

2.7) What are the values of  $A_N$ ,  $B_N$ ,  $C_N$ ,  $S_N$ ,  $D_N$ ,  $E_N$  in the execution of Program 2 on array A from question 2.6? (Give exact numbers.)

Project Part 3:

3.1-3.2) Implement Program 1 and Program 2 from the paper. Assume that the elements to be sorted are positive integers. Given a filename, both programs should read integers to be sorted from the file, sort them, and write sorted integers to an output file. Your programs should also compute how much time was spent on sorting. The computed time should not include time spent on reading and writing data to/from files.

3.3) Write a separate program (let us call it Generator) which will create a file containing a specified number of random integers within certain range. The inputs to this program should include the number of integers you want to be in the file and the lower and upper bounds for the integers. For example, it should be able to create a file with 1000 integers between 100 (inclusive) and 999 (inclusive).

3.4) Generate 10 different input files using your Generator program. Each file should have ten thousand integers in the range from 1 to 10000.

3.5) Sort each of these input files using program 1. Record sorting times. What are the average, minimum and maximum sorting times?

3.6) Sort each of the ten input files using Program 2 with values of M equal to 3, 6, 9, 10, 14, 20. Record sorting time in each run. What are the average, minimum and maximum sorting times for each of these values of M?

3.7) Summarize your results in a table like the following:

Sorting time	Program 1	Program 2					
		M=3	M=6	M=9	M=10	M=14	M=20
Average							
Minimum							
Maximum							

Answer the following questions.

- a. Is sorting time for Program 2 always better than sorting time for Program 1 in your experiments?

- b. What value of M in your experiments gave the best sorting time? Is that value of M the same as the best value of M in Sedgwick's paper?
- c. Is there a big difference between average, minimum and maximum sorting times in your experiments?

## References

- [1] Hoare, C. A. R. "Partition: Algorithm 63," "Quicksort: Algorithm 64," and "Find: Algorithm 65." *Comm. ACM* **4**, 321-322, 1961.
- [2] Hoare, C. A. R. "Quicksort." *Computer J.* **5**, 10-15, 1962.
- [3] Sedgwick, R. [\*Quicksort\*](#). Ph.D. thesis. Stanford Computer Science Report STAN-CS-75-492. Stanford, CA: Stanford University, May 1975.
- [4] Sedgwick, R. "The Analysis of Quicksort Programs." *Acta Informatica* **7**, 327-355, 1977.
- [5] Sedgwick, R. "Implementing Quicksort Programs." *Comm. ACM* **21**, 847-857, 1978. DOI= <http://doi.acm.org/10.1145/359619.359631>.