UDC 004.021
DOI https://doi.org/10.32782/tnv-tech.2023.6.6

CONCEPT OF BUILDING A LIBRARY OF TASKS AND SOLUTIONS, PART 2: SIMPLE SORTING

Paulin O. M. *– Doctor of Technical Sciences,*
Associate Professor at the Department of Software Engineering
National University "Odesa Polytechnic"
ORCID ID: 0000-0002-2210-8317

Komleva N. O. *– Doctor in Engineering, Associate Professor,*
Head at the Department of Software Engineering
National University "Odesa Polytechnic"
ORCID ID: 0000-0001-9627-8530

Nikitchenko M. I. *– Graduate Student*
National University "Odesa Polytechnic"
ORCID ID: 0009-0007-9560-7057

This paper is a continuation of the article "CONCEPT OF BUILDING A LIBRARY OF TASKS AND SOLUTIONS" [1], which is devoted to the concept of building a library of common problems and their solutions in the form of computational processes and macro operations, as well as their models based on Petri nets. The library is a tool for collecting and systematizing various problems, their solutions, and models. The numbering of tasks and solutions is introduced.

It is based on a tree structure that is convenient for both developers and practitioners in the field of computer science. In the previous paper, special attention was paid to the architecture and structure of the library, which is a tree whose nodes store knowledge about specific problems, methods of solving them, and relevant computing processes, which provides a deep understanding of the characteristics of the problem and its solution.

In the previous article, we mentioned the division of the library into an open part accessible to the user and a closed part that is under the control of the developer.

This article describes how to fill the library with two tasks: simple selection sorting and simple exchange. Each task is presented with a detailed solution containing: a task model, a solution method, a computational process (CP) in the form of a verbal description of the algorithm, an algorithm diagram, a list of macro operations, a CP model in the form of a Petri net with a description of the network elements and scenarios for its verification.

To maintain the style of presentation of the problem solution, the article is also supplemented with inserts that lack information for sorting: the problem model and the method of its solution. In addition, the Sorting node of the library contains theoretical information about sorting: what is sorting, types of sorting, their features, and for simple sortings, a table of quality assessment of the sorting process.

Key words: *library of problems and solutions, numbering, sorting by simple selection and exchange, problem model, solution method, verbal description of the algorithm, algorithm diagram, network model of problem solving, macro operation.*

Паулін О. М., Комлева Н. О., Нікітченко М. І. Концепція побудови бібліотеки задач та рішень, частина 2: прості сортування

Ця робота є продовженням статті "CONCEPT OF BUILDING A LIBRARY OF TASKS AND SOLUTIONS" [1], присвяченій концепції побудови бібліотеки поширених задач та їхніх розв'язків у вигляді обчислювальних процесів і макрооперацій, а також їхніх моделей на основі мереж Петрі. Бібліотека є інструментом для збору та систематизації різноманітних задач, їхніх розв'язків і моделей. Вводиться нумерація задач і рішень.

Вона будується на основі деревоподібної структури, зручної як для розробників, так і для практиків у галузі комп'ютерних наук. Особливу увагу в попередній роботі було приділено архітектурі та структурі бібліотеки, яка являє собою дерево, у вузлах якого

зберігаються знання про конкретні задачі, методи їхнього розв'язання та відповідні обчислювальні процеси, що забезпечує глибоке розуміння особливостей задачі та її розв'язання.

У минулій статті згадувалося про поділ бібліотеки на відкриту частину, доступну користувачеві, і закриту частину, що перебуває у віданні розробника.

У цій статті описується наповнення бібліотеки двома завданнями – сортуванням простим вибором і простим обміном. Кожну задачу представлено докладним розв'язком, що містить: модель задачі, метод розв'язання, обчислювальний процес (ОП) у вигляді словесного опису алгоритму, схему алгоритму, список макроопераций, моделі ОП у вигляді мережі Петрі з описом елементів мережі та сценаріїв її верифікації.

Для підтримки стилю представлення розв'язання задачі статтю також доповнюють вставками, яких бракує інформації для сортування: моделлю задачі та методом її розв'язання. Крім того, у вузол "Сортування" бібліотеки вводяться теоретичні відомості про сортування: що таке сортування, види сортувань, їхні особливості, а також для простих сортувань – таблиця оцінок якості процесу сортування.

Ключові слова: *бібліотека задач і розв'язків, нумерація, сортування простим вибором і обміном, модель задачі, метод розв'язання, словесний опис алгоритму, схема алгоритму, мережева модель розв'язання задачі, макрооперація.*

Introduction. This study continues the theme started in the previous paper [1]. In it, the foundations were laid for the creation of an extensive library designed to collect and systematically tize a variety of problems and their solutions. The library covers various aspects of problem solving: problem models, solution methods, computational processes (CPs) and macrooperations (MOs), as well as their models based on Petri nets (CP/MO models). The library is organized hierarchically in the form of a tree, the first 3 levels of which reproduce the classification of problems, and the next one – different aspects of problem solving.

The relevance of the research is determined by the fact that in practice one often spends many times more time on debugging programs than on writing them. To eliminate this imbalance, we propose to shift the burden of the struggle for program quality to an earlier stage of its life cycle – to the stage of building a high-quality CP. In this case, it is proposed to model the CP on the basis of Petri net.

The aim of this study is to improve the quality of the CP by further developing and deepening the functionality of the proposed library.

In order to achieve this goal, the following task was defined:

- Filling the library with two next sorting problems and their detailed solutions by the methods of simple selection and simple exchange.

A detailed description of these solutions will provide a better understanding of the solutions and efficient utilization of library resources.

Main part

Sorting task class

Sorting is the process of rearranging the elements of a one-dimensional array in a certain order. The elements can be numbers, strings, database records, or any other data. Depending on the specific requirements and characteristics of the data, there are many different sorting methods. The numeric attributes of the elements to be sorted are called keys.

Note. We will use the term element as the more common term, always referring to a key.

Sorts can be divided into simple and efficient sorts, as well as sequential and parallel sorts. All sorts implement the Compare-Replace operation for a pair of elements; the difference between the sorting methods lies in the way the pairs are formed.

Simple sorts are easy to understand and implement [2]. They can be effective for small amounts of data or when performance is not critical. The library includes:

insertion sorting, selection sorting and exchange sorting. These are often used for training purposes or when simplicity is more important than performance. Efficient sorts usually provide good performance on large amounts of data. They can be more complex to implement, but provide better performance in the average and worst case. Shell sort, pyramidal sort, and fast sort. which have better asymptotic complexity will be included in the library as efficient sorts. The listed sorts are sequential and only one pair of elements is processed at each step of the CP. Parallel sorts are used to speed up the sorting process of large amounts of data; boosting on multicore or distributed systems. Among them we can mention bitonic sorting, as well as modifications for merge sorting and fast sorting.

In this article, only simple sorts will be considered.

General description of simple sorts

The sorted sequence is divided into 2 subsequences: *ready* and *input*. We introduce the basic notion of "*boundary*" separating the ready subsequence (RSS) and the input subsequence (ISS). We consider the current state of the sorting process and specify the new position of the boundary at the next step of the CP, as well as its initial position.

In the course of sorting by simple insertions, a matching location in the RSS is sought for the current boundary element x. The pair of compared elements is (x, a_{j+1}).

In the case of sorting by simple selection, the current boundary element a_i exchanges places with the minimum element a_k of the ISS. The pair is (a_i, a_k).

In simple exchange sorting, the sequence is presented vertically, with the first element placed at the top. The boundary is set between the 1st and 2nd elements. Nearby elements are compared; they form a pair.

In order to build a convenient navigation through the library, we will introduce the following numbering: Digit.Digit. Latin capital letter. Here the first digit means the class of problems, the second digit means the subclass of problems, and the letter means the solutions of problems.

We propose the following solution structure: a model of the problem, a method of its solution, a verbal description of the algorithm\MO, a PN-model of the CP\PN.

1.1.A. The *combinatorial* subclass *sorting* problem by simple insertions.

A unified representation of the problem solution was proposed above, but the solution file of this problem [1] lacks two solution components: problem model, solution method and algorithm scheme (AS). Let us fill this gap.

Task model

The idea of solving the problem is to find a suitable place for inserting into the current RSS an element of the current ISS located just outside the boundary on the right. The idea is illustrated by the problem model presented in Fig. 1.

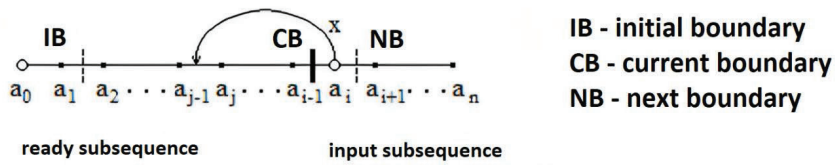

Fig. 1. Model of the sorting problem by simple insertion method

Sorting method by simple insertion

0. The initial boundary is set to the 2nd element of the element sequence.

1. The element $x=a_i$ beyond the right border in the current ISS is placed in a suitable place of the current RSS. This place is determined by the condition $a_{j-1} \le x < a_j$, where j is the number of the current element in the RSS.

2. The boundary between the RSS and the ISS is moved one position to the right. Move to step 1.

The AS of this sorting is constructed by verbal description of the algorithm (VDA); it is shown in Fig. 2.

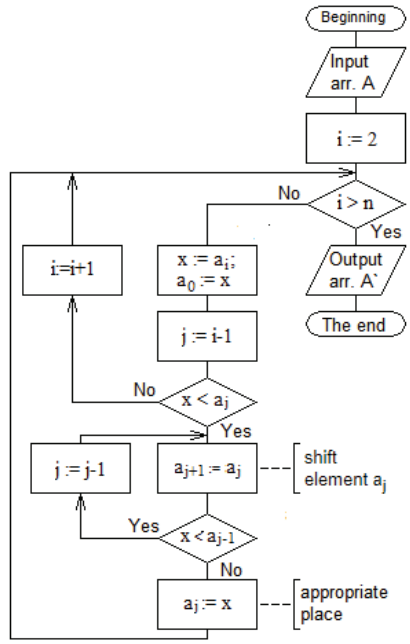

Fig. 2. AS of sorting by inserts

1.1.B. Sorting problem by simple selection method

Task model

Sequence elements are divided into RSS $a_1, a_2, ..., a_{i-1}$ and ISS $a_i, ..., a_n$, which is indicated in Fig. 3. The idea is then used: the first element in order from the ISS is swapped with the smallest ISS element found, after which the boundary is shifted, resulting in that element being in the RSS.

Method for solving the problem

0. The initial boundary is set to the 2nd element of the given sequence of elements.

1. Search for the minimum element in the current ISS and, if necessary, exchange it with its 1st element.

2. Shift the boundary one position to the right. Move to step 1.

Data Structures:

A, A' – sequences of elements given and sorted;

i, j – parameters of cycles, external and internal;

n is the number of keys in the sequence;

k is a special key (element).

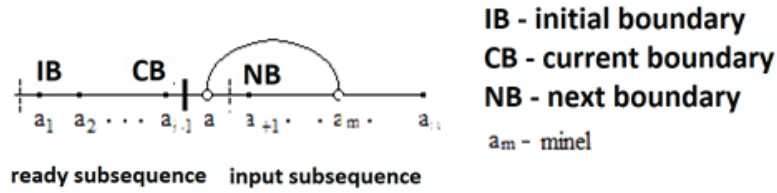

Fig. 3. Model of the sorting problem by simple selection method

A verbal description of the algorithm

1. Input the key sequence A; i := 1.

2. Check i < n? If YES, then set the values k := i, j := i+1 and go to step 3, otherwise go to step. 6.

3. Find the smallest element in the input subsequence a_i, ..., a_n and assign k its index. To do this, perform the following:

- Browse all elements of the subsequence from a_i to a_n.
- Each element a_j, j = i + 1..n, is compared to an element a_k .
- If the element $a_k > a_j$, we assign the value j to k.

4. Increase the value of j by 1. If j <= n, go to step 3. Otherwise, check $i \neq k$? If YES, then a_i and a_k are swapped.

5. i := i + 1. Go to step 2.

6. Output of the sequence A'.

According to the VDA, an AS is constructed; it is shown in Fig. 4. Further, the full PN-model for the CP of sorting by simple selection is constructed (Fig. 5). Here the numbers 0 and 1 near the arc leaving the position denote: 1 – the process continues, 0 – transition to an alternative process.

Tables 1 and 2 summarize the description and purpose of the components for this PN model.

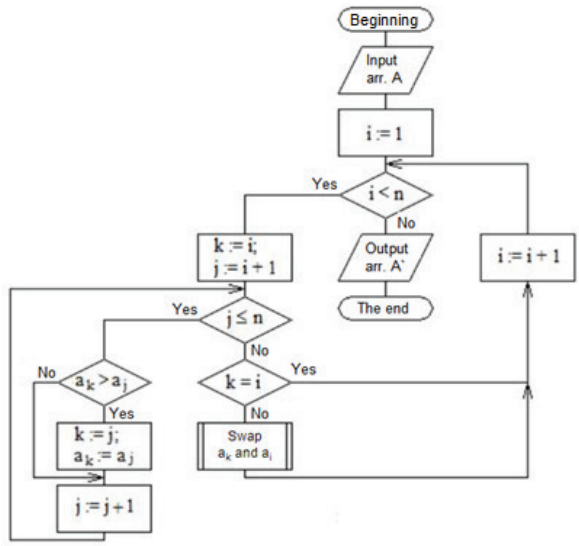

Fig. 4. AS for sorting by simple selection

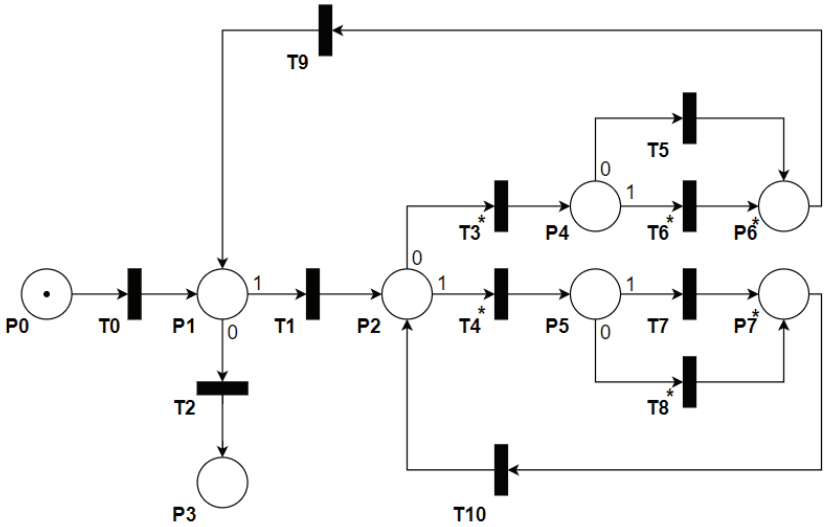

Fig. 5. PN-model of CP for sorting by simple selection

Table 1

Positions and their purpose

Positions	Purposes
p_0	Beginning
p_1	i < n?
p_2	The end
p_3	j <= n?
p_4	k = i?
p_5	$a_k > a_j$?
p^*_6	∅
p^*_7	∅

Table 2

Transitions and their purpose

Positions	Purposes
t_0	Array input, i:= 1;
t_1	output A’
t_2	j := i; k :=j + 1
t^*_3	NOP
t^*_4	NOP
t^*_5	NOP
t_6	k := j
t_7	Swap places a_i and a_k
t_8	j :=j + 1
t_9	i :=i + 1

Based on the structure of the PN, the following transition triggering scenarios can be identified, covering all possible branches/loops of the structure. We have:

1) p0 → t0 → NOTp1 → t2 → p3
2) p1 → t1 → NOTp2 → t3 → NOTp4 → t5 → p6 → t9 →....
3) p1 → t1 → NOTp2 → t3 → p4 → t6 → p6 → t9 →....
4) p2 → t4 → p5→t7 → p7 → t10 →....
5) p2 → t4 → NOTp5→t8 → p7 → t10 →....

1.1.C. Sorting problem by simple exchange

Task model

Fig. 6 shows the sequence elements, which are also divided into RSS *a, $a_{12,}$ ..., a_{i-1}* and ISS a_i, ..., a_n. Unlike the previous variant, here the sorting is performed from right to left. And here the elements are compared in pairs (a_n and a_{n-1}, then a_{n-2} and a_{n-1}, etc.). The smaller element is successively shifted until it is in place of a_i. Then the boundary is shifted so that a_i is in the RSS and the comparison process starts again.

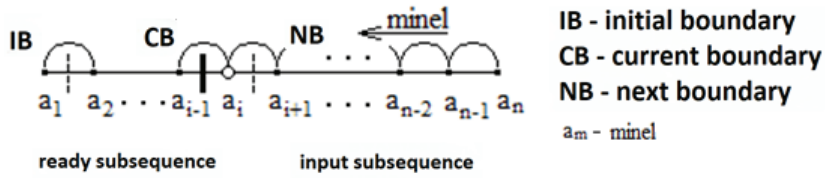

Fig. 6. Model of the sorting problem by simple exchange method

Method for solving the problem

0. The initial boundary is set over the uppermost element of the sequence of elements.

1. Pairs of neighboring elements are formed in the current input subsequence, starting from the bottom; they swap places if necessary. The "lighter" elements "pop up" up to the boundary.

2. Border moves down a position; move to step 1.

Data Structures:

A, A' are sequences of keys given and sorted;

i, j – parameters of cycles, external and internal;

n is the number of keys in the sequence;

A verbal description of the algorithm by a simple exchange

1. Input sequence A; set the value i := 2.

2. Check i > n. If NO, set j := n and go to step 3, otherwise sorting is over.

3. Check j < i. If YES, then i := i+1 and go to step 2.

4. Compare the values of a_j and a_{j+1}. If $a_{j+1} < a_j$, we swap them. Decrease the value of j := j – 1, go to step. 3.

5. Output of the sequence A'.

For this algorithm, the AS is constructed according to its SOA (Fig. 7).

Below we consider a complete PN-model for CP sorting by simple exchange (Fig. 8), constructed by AS. Here the numbers 0 and 1 near the arc leaving the position denote: 1 – the process continues, 0 – transition to an alternative process.

Table 3 and Table 4 summarize the description and purpose of the components for this PN model.

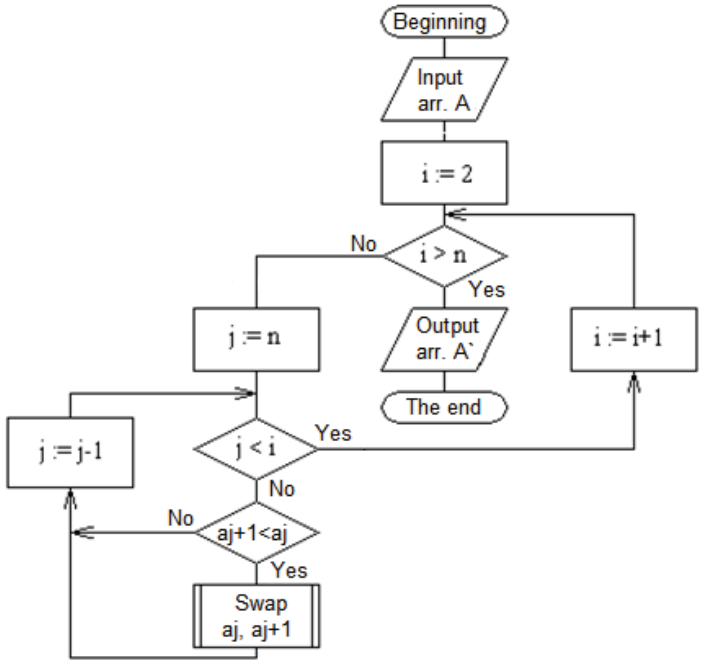

Fig. 7. AS for sorting by simple exchange

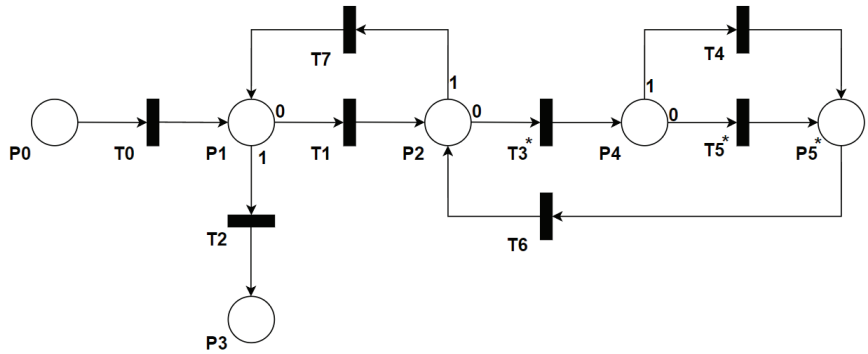

Fig. 8. PN model of PN for sorting by simple exchange

Based on the structure of the PN, the following transition triggering scenarios can be identified, covering all possible branches/loops of the structure. We have:

1) p0 → t0 → p1 → t2 → p3
2) NOTp1 → t1 → p2 → t7 →...
3) p2 →t3 → NOTp4 → t5 → p5 → t6 →…
4) p2 →t3 → p4 → t4 → p5 → t6 →…

Table 3

Positions and their purpose

Positions	Purposes
p_0	Beginning
p_1	i > n?
p_2	The end
p_3	j < i?
p_4	$a_{j+1} < a_j$?
p^*_4	∅

Table 4

Transitions and their purpose

Positions	Purposes
t_0	Array input, i: = 2
t_1	j: = n
t_2	Output A'
t^*_3	NOP
t_4	Swap places a_j and a_{j-1}
t^*_5	NOP
t_6	j:= j-1
t_7	i:= i+1

Conclusion. In this paper we continue filling the library of problems and solutions started in the previous article; two problems were added to the library: sorting by simple selection and simple exchange. At the same time, an internal standard of the library was formed to represent the solution of problems, which includes: the problem model, the solution method, the CP in the form of a verbal description of the algorithm and MO, the algorithm scheme, the CP and MO models in the form of a Petri net with a description of the network elements, and scenarios for its verification. In accordance with this standard, the description of the solution to the problem of sorting by simple inserts was adjusted.

The paper proposes a method of numbering problems and solutions.

The use of Petri nets to model CPs, together with detailed descriptions of problems and solutions, has made the library not only a tool for storing data, but also a valuable resource for analyzing and managing CPs. This approach enhances the quality of programming and enriches learning and practice in computer science.

BIBLIOGRAPHY:

1. Паулін, О. М., Комлева Н.О., Нікітченко, М. І. КОНЦЕПЦІЯ ПОБУДОВИ БІБЛІОТЕКИ ЗАДАЧ ТА РІШЕНЬ // *Таврійський науковий вісник. Серія: Технічні науки*. 2023. №. 5.

2. Wirth N. Algorithms & Data Structures. Pearson Education, Limited, 1986. 288 p.

REFERENCES:

1. Paulin, O. M., Komleva N.O., Nikitchenko, M. I. (2023) CONCEPT OF BUILDING A LIBRARY OF TASKS AND SOLUTIONS // Taurida Scientific Herald. Series: Technical Sciences. No. 5.

2. Wirth N. (1986) Algorithms & Data Structures. Pearson Education, Limited, 288 p.