Test1-Sorting Algorithm

Xubang XIONG
The Hong Kong University of Science and Technology
Hong Kong, China
xxiongag@connect.ust.hk

ABSTRACT

Sorting is necessary and key to practical scenario, like educational management system. However, the existing researches ignore the formulation of sorting problem and there is no effective sorting algorithm. Motivated by this situation, we propose sorting problem and design an algorithm called *Selection Sort* to solve the problem. The algorithm is proved to be correct. And its time complexity is $O(m^2)$, where m is the size of sequence. Eventually, extensive experiments reveal the effectiveness and efficiency of our algorithm.

CCS CONCEPTS

• **Theory of computation → Design and analysis of algorithms**; • **→ Data structures design and analysis**; **Sorting and searching**.

KEYWORDS

computation theory, algorithm design, sorting

1 INTRODUCTION

Recently, designing management systems for ranking students' scores is popular in large educational institutions. Given an unordered sequence of numbers like grades, the systems are able to output the ordered sequence so that teachers can easily rank their students' learning abilities by the sequence. Besides, in process scheduling system in operation system and in commodity price monitoring system, generating ordered sequences based on the unordered ones is essential.

In the applications mentioned above, sorting is the key. However, there is no existing research concentrating on sorting problem, which aims to arrange an unordered sequence into an ordered sequence. In this paper, we present a sorting algorithm call *Selection Sort* to solve the problem. It iteratively chooses the maximum value from unordered sequence and switches the position of the value and that related to the head of unordered sequence.

To illustrate how *Selection Sort* works, we present a motivating example as follows. Given a series of students' grades $G = \{A = 76, B = 64, C = 99, D = 88, E = 95\}$, teachers want to rank their learning abilities by grade. Teachers can employ *Selection Sort* to arrange the unordered grade sequence. At the beginning of the algorithm, it chooses the maximum value $C = 99$, then switches the position of $C = 99$ and $A = 76$, which is the first element of unordered sequence. After that, the algorithm repeats the above steps, until teachers get an ordered grade sequence $G' = \{C = 99, E = 95, D = 88, A = 76, B = 64\}$.

The key contributions of our work are summarized in the following:

- To the best of our knowledge, this is the first work on sorting problem. We formulate the sorting problem and propose a metric to measure whether sorting algorithm works.
- We design a sorting algorithm: *Selection Sort*, which is proved to solve sorting problem effectively and efficiently.
- Extensive simulations have validated the viability of the proposed algorithm and the analysis of execution time gives guidance for utilizing the algorithm.

The rest of the paper is organized as follows. Section 2 reviews related work. The proposed sorting algorithm is formulated in Section 3. Section 4 describes the design of our algorithm. Finally, Section 5 shows the performance of our algorithm and Section 6 concludes this paper.

2 RELATED WORK

There is no existing related work because we are the first one to research the sorting problem.

3 PROBLEM DEFINITIONS

Problem 1. *(**Sorting Problem**) Given a sequence of unordered numbers $N_u = \{n_1, n_2, ..., n_m\}$, we aim to generate a sequence of numbers $N_d = \{n'_1, n'_2, ..., n'_m\}$ in descending order.*

Definition 1. *(**Accuracy**) Generally, we set N_d as ground truth. After conducting sorting algorithm, we can generate a sequence, denoted by $N_o = \{n''_1, n''_2, ..., n''_m\}$. Then, we use I_i indicates whether the i-th element n''_i of the output is equal to the ground truth n'_i. If they are equal, we set $I_i = 1$, otherwise we set $I_i = 0$. The accuracy of the output of algorithm can be defined as:*

$$ACC = \frac{\sum_{i=0}^{m-1} I_i}{m}.$$

Note that *ACC* can be used to measure whether sorting algorithm works. If the algorithm works correctly, *ACC* will be always 100%.

Taking score ranking as an example, teachers get a lot of students' scores, which are unordered. And teachers would like to rank these scores in descending order so as to select the outstanding students. After using our sorting algorithm, teachers can obtain a series of scores within which outstanding students rank at the top.

Notation	Description
N_u	a sequence of unordered numbers
N_d	a sequence of numbers in descending order
N_o	the output of sorting algorithm
m	the size of sequence

Table 1: Frequently used notations

Algorithm 1 Selection Sort

Input: $N_u = \{n_1, n_2, ..., n_m\}$
Output: $N_d = \{n'_1, n'_2, ..., n'_m\}$
1: $N_d \leftarrow N_u$
2: **if** $m \leq 1$ **then** ▷ There is no need to sort the sequence.
3: **return** N_d
4: **end if**
5: **for** $i \leftarrow 0$ to $m-2$ **do**
6: $maxIdx \leftarrow i$
7: **for** $j \leftarrow i+1$ to $m-1$ **do**
8: **if** $N_d[j] > N_d[maxIdx]$ **then**
9: $maxIdx \leftarrow j$
10: **end if**
11: **end for**
12: **if** $maxIdx \neq i$ **then**
13: $temp \leftarrow N_d[i]$
14: $N_d[i] \leftarrow N_d[maxIdx]$
15: $N_d[maxIdx] \leftarrow temp$
16: **end if**
17: **end for**
18: **return** N_d

4 ALGORITHM

4.1 Algorithm Design

To solve the sorting problem, we propose *Selection Sort* algorithm. *Selection Sort* selects the maximum value from unordered sequence. After being moved to the rear of ordered sequence, the value is as an element of ordered sequence. The algorithm repeats the above steps until all elements are in descending order.

Specifically, *Selection Sort* only sorts the sequence which contains more than one element (line 2-4 in Algorithm 1). Then in line 6-11, the proposed algorithm chooses the index that is related to the maximum value in unordered sequence. In line 12-16, if the index is not related to the rear of ordered sequence, the proposed algorithm will swap the maximum value for the value in rear of ordered sequence.

In the following, we use an example to illustrate how *Selection Sort* works.

Example 1: Considering five students' grades $G = \{A = 76, B = 64, C = 99, D = 88, E = 95\}$, teachers want to rank their grades and reward the outstanding students(top 2) with scholarships, using *Selection Sort*. Then, the proposed algorithm works as follows.

- $G' = G = \{A = 76, B = 64, C = 99, D = 88, E = 95\}$.
- $i = 0, maxIdx = 2, G'[maxIdx] = 99$, thus $G' = \{C = 99, A = 76, B = 64, D = 88, E = 95\}$.
- $i = 1, maxIdx = 4, G'[maxIdx] = 95$, thus $G' = \{C = 99, E = 95, A = 76, B = 64, D = 88\}$.
- $i = 2, maxIdx = 4, G'[maxIdx] = 88$, thus $G' = \{C = 99, E = 95, D = 88, A = 76, B = 64\}$.
- $i = 3, maxIdx = 3, G'[maxIdx] = 99$, then we do nothing, thus $G' = \{C = 99, E = 95, D = 88, A = 76, B = 64\}$.
- The output is $G' = \{C = 99, E = 95, D = 88, A = 76, B = 64\}$. C and E will be awarded with the scholarships.

4.2 Analysis

Firstly, we prove that *Selection Sort* works correctly. Then, we prove the complexity of the proposed algorithm. Finally, we discuss the unstableness of it.

Theorem 1. *The* Selection Sort *algorithm can solve the sorting problem correctly.*

Proof. In Algorithm 1, each iteration of the outer loop(line 5-17) indicate that the maximum value in unordered sequence was moved to the rear of ordered sequence. The value is larger than any following element and less than any previous element. Thus, the order of this value is correct. Then ordered sequence can always keep in descending order. □

Lemma 1. *The number of operation of swapping data is* $O(m)$.

Proof. According to Algorithm 1, the operation from line 12 to 16 will be executed 0 to $m-1$ times. Hence, the average operation number is $\frac{m-1}{2}$. □

Lemma 2. *The number of operation of comparing data is* $O(m^2)$.

Proof. The number of operation of comparing data is fixed. The total number of it is

$$TN = (m-1) + (m-2) + ... + 1 = \frac{m \times (m-1)}{2}.$$

□

Theorem 2. *The time complexity of* Selection Sorting *is* $O(m^2)$.

Proof. Given *Lemma 1* and *Lemma 2*, this theorem is proven. □

Theorem 3. Selection Sort *is an unstable sorting algorithm.*

Proof. Given a series of students grades $\{A = 85, B = 98, C = 85, D = 82, E = 99\}$. When we use *Selection Sort*, it firstly finds the maximum value $E = 99$ and switch the position of E and A. In this way, after sorting, A is behind C while A is in front of C at the beginning. □

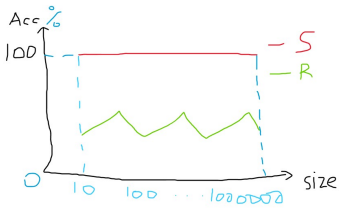
5 EXPERIMENT

5.1 Settings

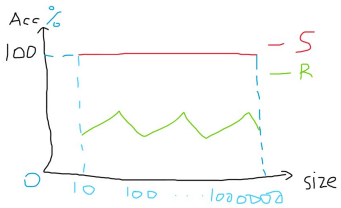
Our experiments were conducted with C/C++ implementation on a machine with 2.40GHz CPU and 16GB RAM.

Datasets. We conducted experiments on synthetic and real datasets. For synthetic datasets, we randomly generate 1000000 number sequences in descending order, of which the sizes are not equal. We set these sequences as the ground truth. Then we shuffle the order of these sequences and set them as the input of our proposed algorithm. For real datasets, we use the students' transcripts in UST. There are 9999999 records of students' transcripts, which contain 10 grades of different subjects.

Algorithm. We compare *Selection Sort* with *Random*. Because there is no existing related sorting algorithm, we applied the random algorithm to sorting problem. For *Random*, it iteratively chooses two indexes, and swaps the related values.

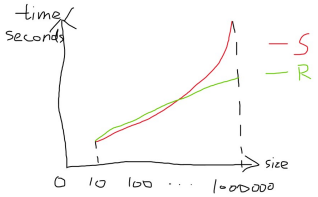


(a) Real Dataset

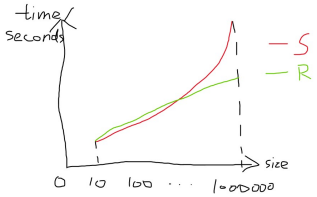


(b) Synthenic Datasets

Figure 1: Accuracy



(a) Real Dataset



(b) Synthenic Datasets

Figure 2: Execution Time

Parameters. We evaluated the algorithms by varying: (1) The size of dataset m. In this experiment, we set it to 10, 100, ..., 1000000.

Measurement. We evaluated the performances of the algorithms by two measurements: (1) *Execution time*; (2) *Match accuracy*.

Unless stated explicitly, we conducted the algorithms 50 times with different measurements. And we employed the averages as the final results. We stopped an algorithm if its total execution time was more than 10^4 seconds.

5.2 Results

For *Accuracy*, we can observe that *Selection Sort* outperforms *Random*. And the accuracy of *Selection* is 100% varying sequence sizes. It also proves the correctness of our algorithm. For *Execution time*, the figure 2 shows that the execution time of both algorithms increases with size. Besides, the time of *Selection Sort* increases dramatically, indicating that its complexity is $O(m^2)$. When the sequence size is less than a certain value, *Selection Sort* costs less time than *Random*.

To sum up, *Selection Sort* is the state-of-the-art, and its execution time is efficient.

6 CONCLUSION

In this paper, we presented the sorting problem. In order to solve the problem, we proposed *Selection Sort*, which can be proved to be $O(m^2)$ theoretically, where m is the size of sequence. We conducted extensive experiments in both synthetic and real datasets. They both showed that our algorithm is effective and efficient. As for future work, we consider that the complexity of sorting algorithm can be reduced. Then, we can expand our 1-dimension algorithm to higher dimensions algorithm. Last but not least, we can design a stable sorting algorithm based on *Selection Sort*.

7 TEST FOR CITATIONS

Note that this reference[1] is only used to test.

REFERENCES

[1] Yuanfeng Song, Xuefang Zhao, Raymond Chi-Wing Wong, and Di Jiang. Rgvisnet: A hybrid retrieval-generation neural framework towards automatic data visualization generation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 1646–1655, New York, NY, USA, 2022. Association for Computing Machinery.