Teaching Exploration of Algorithm Time Complexity Analysis

Xirimo Bao[1,*], Chunmei Ning[2,a]

[1]School of Intelligence Science, Hohhot Minzu College, Hohhot 010051, China
[2]Library of College, Hohhot Minzu College, Hohhot 010051, China

*xirimo_bao@sina.com, a378303856@qq.com

Abstract. The analysis of algorithm time complexity occupies a crucial position in the teaching of the data structure course for computer majors in undergraduate colleges and universities. By introducing such guiding thoughts as "concept clarification" and "problem classification and solving" into the teaching of algorithm time complexity analysis, the authors has overcome many shortcomings in conventional teaching, such as lack of concept clarity, over-emphasis on theoretical teaching and neglect of ability cultivation, and achieved excellent teaching effects in teaching practice.

Keywords: data structure; algorithm; algorithm analysis; time complexity
Classification Number: G712 Document Code: A

1 Introduction

Data Structures is a foundational course for the vast majority of computer-related majors. Whether they are traditional majors or emerging disciplines, Data Structures occupies a pivotal position in their respective curriculum systems. The concept and analysis of algorithm time complexity are key and difficult points of the Data Structures course. The teaching effect of this content directly impacts the overall quality of the Data Structures course, and even exerts a far-reaching influence on the teaching of subsequent specialized courses. We attempt to carry out teaching reforms on the concept and analysis of algorithm time complexity, and proposes an innovative teaching model with the aim of achieving excellent teaching effects.

2 Conventional Teaching of Algorithm Time Complexity Analysis

In the conventional teaching mode, after denying the method of directly measuring the execution time of an algorithm, it is usually pointed out that the number of repeated executions of the basic operations of the algorithm, denoted as f(n) (where n represents the problem size), should be used as the measure of the algorithm's execu-

A. Y. M. A. Islam et al. (eds.), *Proceedings of the 2025 International Conference on Educational Technology and Management Information Systems (ETMIS 2025)*, Advances in Computer Science Research 129,
https://doi.org/10.2991/978-94-6239-630-2_14

tion time. Then, the definition of the asymptotic time complexity of the algorithm (referred to as time complexity for short) is directly introduced:

$$T(n) = O(f(n))$$

It is also stated that T(n) indicates that as the problem size n increases, the growth rate of the algorithm's execution time is the same as that of f(n) [1-3]. This way of introducing the concept of algorithm time complexity has a large leap, unreasonable connection, and is obscure and difficult to understand, which is not conducive to classroom teaching. In teaching practice, there is an urgent need for an easy-to-understand way of introducing the concept of algorithm time complexity with a reasonable connection between the previous and subsequent content. This will help students understand the origin and development of the concept, know not only the facts but also the reasons behind them, and form a complete knowledge system.

In addition, the conventional teaching mode usually over-emphasizes the imparting of theoretical knowledge and neglects the practical teaching links of time complexity analysis [4-8]. There is an urgent need to increase practical teaching links in the teaching process so that students can master the methods and skills of algorithm time complexity analysis, thereby improving their ability to analyze and solve practical problems.

3 Clarification of the Concept of Algorithm Time Complexity

We guide students to understand the concept of time complexity and the method of time complexity analysis through the following ideas and steps:

(1) The purpose of algorithm time complexity analysis is to measure the execution efficiency of an algorithm, that is, the execution time of the algorithm. However, it is not advisable to directly convert the algorithm into a program and run it to measure its execution time, as this is not conducive to comparing the complexity between different algorithms. This is because many factors can affect the execution time of an algorithm. The same algorithm will have different execution times on different machines and in the compilation environments of different languages.

(2) A feasible idea is to assume that the execution time of each statement is the same. Under this premise, the total number of statements executed by the algorithm is used as the measure of the algorithm's time complexity. For the purpose of comparing different algorithms, the function representing the total number of statements executed by the algorithm is converted into a more intuitive and unified expression form.

(3) Specific implementation methods of the above idea:

1) Provide the expression of the frequency function, which takes the problem size n as the independent variable and represents the total number of statements executed by the algorithm, such as $f(n) = 4n^3+100n^2+500$. If the value of f(n) is affected by the initial state of the input data, the case where f(n) is the largest should be considered. That is, the worst-case time complexity is selected among the best-case, average-case, and worst-case time complexities.

2) Select the term in the frequency function f(n) whose value changes the fastest with the increase of the problem size n. Adjust the coefficient of this term to 1 and then enclose it in the big O notation. For example, if the statement frequency function of an algorithm is $f(n) = 4n^3+100n^2+500$, then the time complexity of the algorithm is $O(n^3)$. The reason for selecting the term that changes the fastest with n and adjusting its coefficient to 1 is as follows: The comparison of the time complexity between different algorithms is only meaningful when the problem size n is very large. When n is very large, the term that changes the fastest with n determines the value of f(n), and the influence of the constant coefficient on the value of this term can be ignored.

(4) The above conventional steps can be slightly simplified. The statement that is executed the most times in an algorithm is called the basic operation. In the above steps, in fact, it is only necessary to provide the expression of the frequency function f(n) representing the number of executions of the basic operation, where n still represents the problem size, and the subsequent steps remain unchanged.

4 Problem Classification and Solving in Ability Cultivation

In the teaching practice of time complexity analysis, we has abandoned the approach of only focusing on theory, strengthened the practical teaching links, summarized various types of exercises on time complexity analysis, and taught students the characteristics and analysis methods of each type of exercise. This aims to enhance the cultivation of students' ability to solve practical problems.

4.1 Problem Type I

(1) characteristics of the type

The frequency function f(n) of the basic operation, which is expressed in terms of the problem size n, can be directly given.

(2) analysis method

Provide the expression of f(n), select the term in the sum terms of f(n) that changes the fastest with the problem size n, adjust its coefficient to 1, and then enclose it in the big O notation.

(3) example

```
for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
        ++x;
```

(4) analysis of the example

The basic operation of this algorithm is ++x, and its frequency function is $f(n) = n^2$. Therefore, the time complexity of the algorithm is $O(n^2)$.

4.2 Problem Type II

(1) characteristics of the type

The basic operation is located in a loop body. In addition to the problem size n, the conditional expression that controls the loop contains another variable, and the value

of this variable changes within the loop body.

(2) analysis method

For the other variable i (excluding n) in the conditional expression, consider the values of variable i when the loop body is executed for the 1st, 2nd, 3rd, ..., and finally the t-th time in sequence. Based on the pattern, find the relational expression between t and i, and then derive the function f(n) representing the number of executions of the loop body (i.e., t) by combining with the conditional expression.

(3) example 1

```
void fun(int n)
{    int i = 1;
     while(i <= n)
          i = i * 2;
}
```

(4) analysis of example 1

The basic operation i = i*2 is located in a while loop. The conditional expression of the loop i <= n contains another variable i (excluding the problem size n), and the value of i is constantly adjusted within the loop body. Therefore, this example conforms to the characteristics of type II. The values of variable i when the loop body is executed for the 1st, 2nd, 3rd, and 4th times are 1, 2, 4, and 8 respectively. By inferring according to this pattern, the value of variable i when the loop body is executed for the last time (the t-th time) is 2^{t-1}. Substitute $i = 2^{t-1}$ into the loop conditional expression i <= n, we get $t <= \log_2 n+1$. Since t is an integer, its maximum possible value is $t = \lfloor \log_2 n \rfloor + 1$, that is, $f(n) = \lfloor \log_2 n \rfloor + 1 \approx \log_2 n + 1$. Therefore, the time complexity of the algorithm is $O(\log_2 n)$.

(5) example 2

```
void fun(int n)
{    int i = 1, j = 0;
     while(i + j <= n)
     {    if(i > j) ++j;
          else++i;
     }
}
```

(6) analysis of example 2

The basic operation is the if...else... statement in the loop body. If we regard i+j as a variable, it is easy to see that this example conforms to the characteristics of type II. The values of variable i+j when the loop body is executed for the 1st, 2nd, 3rd, and 4th times are 1, 2, 3 and 4 respectively. By inferring according to this pattern, the value of i+j when the loop body is executed for the last time (the t-th time) is t. Substitute i+j=t into the loop conditional expression i+j <= n, we get t <= n. Therefore, f(n) = n, and the time complexity of the algorithm is O(n).

4.3 Problem Type III

(1) characteristics of the type

The basic operation is in nested multi-level loops, and the control variables of each level of loops interact with each other.

(2) analysis method

From the outer loop to the inner loop, consider the possible values of the control variables of each level of loops in sequence. On the basis of summarizing the patterns, provide the expression of the frequency function f(n) of the basic operation.

(3) example

```
for(i = 1; i <= n; i++)
    for(j = 1; j <= i; j++)
        for(k = 1; k <= j; k++)
            x++;
```

(4) analysis of the example

The basic operation is x++. The number of executions of the basic operation, which is counted based on the possible values of the loop control variables i and j, is shown in the following table 1:

Table 1. Statistics of the Number of Executions of the Basic Operation

Possible Values of i	Corresponding Values of j	Number of Executions of the Basic Operation
1	1	1
2	1,2	1+2
3	1,2,3	1+2+3
⋮	⋮	⋮
n	1,2,3,…,n	1+2+3+…+n

Therefore, the frequency function of the basic operation is:

$$f(n) = 1+(1+2)+(1+2+3)+\ldots+(1+2+\ldots+n) = n(n+1)/4+n(n+1)(2n+1)/12$$

Thus, the time complexity of the algorithm is $O(n^3)$.

4.4 Problem Type IV

(1) characteristics of the type

Recursive algorithm.

(2) analysis method

Analyze the recursive relationship between f(n) and f(n-1), and then derive the expression of f(n) based on this relationship. The time complexity of a recursive algorithm is usually of linear order O(n), where n represents the problem size.

(3) example

```
int fact(int n)
{    if(n <= 1) return 1;
     return n * fact(n - 1);
}
```

(4) analysis of the example

Assume that the total number of statements executed by this recursive function is f(n), then:

$$f(n) = f(n-1)+1=f(n-2)+2=\ldots=f(2)+n-2=f(1)+(n-1)=n$$

Therefore, the time complexity of the algorithm is O(n).

5 Analysis and Verification of the Teaching Model Reform

At the theoretical level, this teaching reform is underpinned by well-established educational frameworks. From the perspective of constructivism, the "problem classification and solving" model guides students to conduct classified analysis of practical problems, prompting them to take the initiative in constructing knowledge systems. This transforms the passive reception of theoretical knowledge into active exploration of problem-solving strategies, which aligns with the core constructivist proposition of learner-centered active learning. In the dimension of Bloom's taxonomy of educational objectives, the reform breaks through the limitations of low-order cognitive skills, such as memorizing time complexity calculation formulas. Instead, it focuses on cultivating students' higher-order thinking abilities, including analysis(classification of algorithm complexity analysis problems), evaluation (selection of appropriate complexity analysis methods), and creation (implementation of specific complexity analysis processes for particular scenarios). This effectively addresses the competence-oriented teaching and training requirements.

The effectiveness of this reform plan was verified through quantitative evaluation. The reform plan was implemented in two undergraduate classes majoring in computer science and technology that adopted the hybrid flipped classroom model (students completed basic concept learning via online modules before class, engaged in problem-oriented discussions during class, and conducted hands-on training after class). All participating students had already mastered programming fundamentals (C/C++ languages) and basic data structure concepts through prerequisite courses, which ensured the homogeneity of the teaching context.

Data from the pre-test and post-test showed that the average score of students in the dimension of time complexity concept mastery increased from 63.1 ± 6.2 (out of 100) to 80.9 ± 7.6, with a growth rate of 28.21%. In the dimension of practical problem analysis ability, the average score rose from 62.7 ± 10.3 to 85.3 ± 9.2, representing a growth rate of 36.04%. Compared with the control group (students receiving traditional lecture-based teaching), the experimental group demonstrated significant advantages in both the final theoretical assessment (84.1 ± 8.4 vs. 78.3 ± 6.5) and the practical algorithm complexity analysis assessment (average accuracy of complexity analysis: 86.4% vs. 77.6%). The above data quantitatively verify the superiority of the proposed teaching method over the traditional model.

6 Conclusion and Prospect

In the conventional teaching of algorithm time complexity analysis, when introducing the abstract concept of time complexity, insufficient attention is paid to the foreshadowing and extension of the content. In the teaching of time complexity analysis,

we take "concept clarification" as the guiding ideology, follows the teaching and cognitive laws, and guides students step by step. This enables us to teach students the concept of time complexity as well as its causes and consequences, thereby helping students gain a more comprehensive and in-depth understanding of the concept of algorithm time complexity.

At the same time, we have reformed the conventional teaching mode, which neglects the cultivation of students' abilities and only ends with a few time complexity analysis cases to consolidate the concept of time complexity. Through exploration, on the premise of increasing the proportion of practical teaching links, we have introduced the idea of "problem classification and solving", comprehensively classified and summarized various common problems of time complexity analysis, and taught students different analysis methods for different types of problems. This has achieved the combination of theory and practice, realized the transformation from a knowledge-oriented classroom to an ability-oriented classroom, and cultivated students' ability to analyze and solve practical problems.

Looking ahead, future research will focus on two directions. First, expand the application scope of this teaching method to more diverse student groups to verify its universality. Second, deepen the classification system for the time complexity analysis of complex algorithms with multi-layer control structures, and develop targeted teaching cases and evaluation tools. In addition, further explore the integrated application of intelligent teaching tools to enhance the interactivity and efficiency of the teaching process, and continuously optimize the reform model of algorithm time complexity analysis teaching.

References

1. Yan Weimin, Wu Weimin. Data Structures (C Language Edition)[M]. Beijing: Tsinghua University Press, 2019.
2. Yan Weimin, Wu Weimin, Mi Ning, et al. Data Structures Exercise Collection (C Language Edition)[M]. Beijing: Tsinghua University Press, 2019.
3. Zhou Xingni. A New Perspective on Data Structures and Algorithm Analysis (2nd Edition)[M]. Beijing: Publishing House of Electronics Industry, 2021.
4. Hu Hui, Deng Anyuan, Min Juanjuan. Practical Data Structures and Algorithms (2nd Edition)[M]. Beijing: Beijing University of Posts and Telecommunications Press, 2022.
5. Hong Yunguo. Data Structures (C Language Edition)[M]. Shanghai: Shanghai Jiao Tong University Press, 2023.
6. Li Chunbao, Kuang Zhiqiang, Jiang Lin. Data Structure Tutorial (C++ Language Description)[M]. Beijing: Tsinghua University Press, 2021.
7. Lü Yunxiang, Guo Yingmei, Meng Yao. Data Structures: Python Language Description[M]. Beijing: China Machine Press, 2023.
8. Chen Chaoxiang. Data Structures (C Language Edition) (2nd Edition)[M]. Beijing: Peking University Press, 2024.