RAVIC: Runtime Analysis Visualizer for Introductory Courses

FirstName Surname[†]
Department Name
Institution/University Name
City State Country
email@email.com

FirstName Surname
Department Name
Institution/University Name
City State Country
email@email.com

FirstName Surname
Department Name
Institution/University Name
City State Country
email@email.com

ABSTRACT

As the global economy's demand for computer applications soars, the demand for well-trained computer science (CS) professionals is rising. This trend fuels a surge in CS undergraduate degree production which, in turn, puts a strain on departmental resources. In addition, CS departments have to contend with high dropout and failure rates, fragile learning and bimodal outcomes [1]. Thus, the need for effective teaching techniques in CS education is greater than ever.

Some concepts and skills in CS are very difficult to acquire such as analyzing the running time of short pieces of code [2], and particularly, computing the running time of loops. Most instructors use a trace-sum-approximate flow chart described here [2] to teach students how to analyze the running time of code. The two most important skills in this analysis process are the tracing of code and the approximation of the sum of different number patterns, e.g. the sum of the linear number pattern 1+2+3+...+n is equal to $\frac{n(n+1)}{2}$ and it is approximated to $O(n^2)$. Novice programmers struggle with both these integral skills [3, 4].

Visual systems have been recognized to improve students' understanding of these concepts [4]. Several tools have been developed to assist students with running time analysis. Existing tools are targeted at students with more advanced training, such as a background in advanced data structures (AAV [4]) or experience with code tracing and approximation of the sum of different number patterns (Compigorithm [5]).

Motivation and Proposed Design

However, none of the existing tools teach students how to compute the running time at the introductory level, and this gap presented a unique opportunity for us. An effective runtime analysis visualization system for introductory courses must depict the tracing as well as the approximation of the sum. We propose a design in which the depiction of the loop tracing for runtime analysis is done through the use of a table, where the columns represent loop variables and each row represents one iteration of the loop. More specifically, it tabulates the operations in the order they happen during the execution of the loop (Figure 1).The table is displayed alongside the program. The current line of code being executed is highlighted while the table information is updated. At the end of the execution, the number of operations is determined by summing all the operations in the table column-by-column from the right-most side to the left (Figure 2). The student controls the visualization with the use of navigation arrows to go back and forth. Instructors use a similar technique in the tracing of loops, but this approach allows students to proceed at their own pace. The tool is currently being developed as an extension to VSCode.

REFERENCES

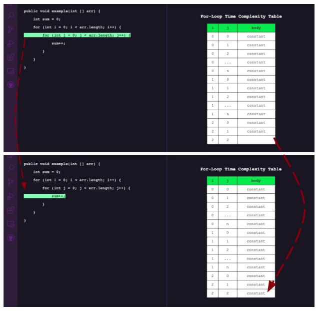
[1] Fincher, S., & Robins, A. (Eds.). (2019). The Cambridge Handbook of Computing Education Research (Cambridge Handbooks in Psychology). *Cambridge: Cambridge University Press.* doi:10.1017/9781108654555

[2] Ibrahim Albluwi et al. 2021. Novice Difficulties with Analyzing the Running Time of Short Pieces of Code. *ACE '21*. ACM, New York, NY, USA.

[3] Kathryn Cunningham et al. 2017. Using Tracing and Sketching to Solve Programming Problems: Replicating and Extending an Analysis of What Students Draw. *ICER '17*. ACM, New York, NY, USA.

[4] Mohammed F. Farghally et al. 2017. Evaluating the Effectiveness of Algorithm Analysis Visualizations. *SIGCSE '17*. ACM, New York, NY, USA.

[5] Rebecca Smith et al. 2020. Compigorithm: An Interactive Tool for Guided Practice of Complexity Analysis. *ITiCSE '20*. ACM, New York, NY, USA.



```
public void example(int [] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr.length; j++) {
            sum++;
        }
    }
}
```

For-Loop Time Complexity Table

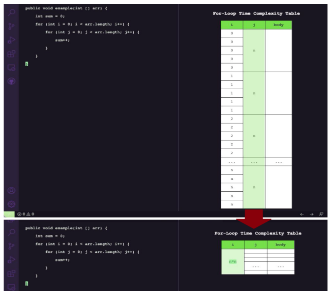
i	j	body
0	0	constant
0	1	constant
0	2	constant
0	...	constant
0	n	constant
1	0	constant
1	1	constant
1	2	constant
1	...	constant
1	n	constant
2	0	constant
2	1	constant
2	2	

```
public void example(int [] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr.length; j++) {
            sum++;
        }
    }
}
```

For-Loop Time Complexity Table

i	j	body
0	0	constant
0	1	constant
0	2	constant
0	...	constant
0	n	constant
1	0	constant
1	1	constant
1	2	constant
1	...	constant
1	n	constant
2	0	constant
2	1	constant
2	2	constant

Figure 1: The tracing stage



```
public void example(int [] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr.length; j++) {
            sum++;
        }
    }
}
```

For-Loop Time Complexity Table

i	j	body
0	n	
0		
0		
0		
0		
1	n	
1		
1		
1		
1		
2	n	
2		
2		
2		
2		
...	...	...
n	n	
n		
n		
n		
n		

```
public void example(int [] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr.length; j++) {
            sum++;
        }
    }
}
```

For-Loop Time Complexity Table

i	j	body
n*n		
	...	...

Figure 2: the summation stage

[*] Article Title Footnote needs to be captured as Title Note

[†] Author Footnote to be captured as Author Note

WOODSTOCK'18, June, 2018, El Paso, Texas USA

978-1-4503-0000-0/18/06...$15.00
https://doi.org/10.1145/1234567890