Athens Journal of Sciences- Volume 2, Issue 3 – Pages 177-186

TIARA Tutor for Time Efficiency Analysis of Recursive Algorithms

By Irena Pevac[*1]

TIARA tutor is an enhanced version of the ESRATEA software developed to assist students to learn time performance analysis of recursive algorithms. The new features include an option for learners to use practice problems in sequential order in addition to the option where problems are randomly selected by the tutoring software. Sequential order is more desirable in the learning phase, while random selection of the problems is more appropriate when students want to test the level of their knowledge after practicing time efficiency analysis in sequential mode. Visualization examples for problems are updated and use a simpler template approach. Practice examples are extended to include time efficiency analysis of more challenging problems, such as generating all permutations for the given set, generating power-set for a given set, computing n-th Fibonacci number as well as analysis of some other examples

Introduction

An algorithms course is typically difficult to teach because it requires students to have a good working knowledge of data structures, discrete mathematics and calculus. The typical course starts with a review of data structures and math preliminaries, and continues with formal analysis of time performance, discussed by introducing o, O, Θ, Ω, and ω notation. Most textbooks (Baase & Van Gelder, 2000, Cormen et al., 2007, Levitin, 2012, McConnell, 2008, Neapolitan, 2014, Pothering & Naps, 2002) provide a table for basic functions such as logarithmic log n, linear n, n log n, quadratic n^2, cubic n^3, exponential a^n and factorial n!. For each function, values are provided for different values of n, and most often for geometric series such as 10, 10^2,…, 10^6 to illustrate the difference in growth. In addition, textbooks provide summation formulas for basic series such as sum of first n integers, sum of first n squares, sum of first n cubes, sum of first n powers of 2, sum of first n powers of d, etc. A few examples of formal time performance analysis of iterative and recursive algorithms are usually provided next.

It has been the authors' experience that such preparation for most students is not sufficient. When students have been tested on their ability to perform

[*]Professor, Central Connecticut State University, USA.
[1]Dedicated to the memory of Professor Djuro Kurepa

https://doi.org/10.30958/ajs.2-3-2 doi=10.30958/ajs.2-3-2

formal time performance analysis of simple algorithms given the above foundation, students still typically struggled with the task.

Problems can be implemented with either iterative or recursive algorithms. Time performance analysis of iterative algorithms is usually easier. It requires: 1) selecting the problem size n; 2) selecting the basic operation; 3) establishing a series that specifies the number of times the basic operation is performed as a function of input size n; and 4) calculating the sum. The time performance analysis of recursive algorithms has proven to be more challenging. It also requires steps 1) and 2) as described above, but after that it requires: 3) establishing recurrences by providing base case and recursive step; and 4) solving the recurrences.

One of the reasons why recursive algorithms time analysis is more difficult to learn is the implicit nature of the recursive approach. In addition, learners are often confused by the two types of recurrence relations. The first type of these relations is recurrences used to specify how to obtain the result of the original problem in terms of one or more sub-problems. These recurrences are used to write the code to solve the problem with a recursive algorithm. The second type of these relations is recurrences used in time analysis. They specify the number of times the basic operation is performed for a problem of size n, denoted as T(n), and depends on: 1) the number of times the basic operation is performed in one or more recursive sub-problems (denoted T(n-1), T(n/2) or similar); and 2) the time f(n), which denotes the number of times the basic operation is performed in the non-recursive part of the code, performed in addition to recursive invocation(s).

One obvious way to improve a learners' knowledge is to provide more examples of time performance analysis using simpler algorithms of approximately the same difficulty level as those given on exams. Unfortunately, spending more class time practicing time efficiency analysis leaves less time for covering different algorithm design techniques and introducing new algorithms of particular domain types.

The tutoring software provided to students has proven to be an effective solution for the problem. TIARA tutor has a wide selection of practice examples with complete time analysis performed. Learners can practice as much as they need based on their background knowledge. Using the software does not take any additional class time since practice is performed outside of class. The tutor is free for students enrolled in an Algorithms course and is available on request to instructors who would like to use it in an Algorithms course taught in a flipped classroom style. At our school, students can download the software as a jar file from the blackboard learning platform, and install it on their own computer, allowing them to practice when convenient for them.

Various researchers and educators have developed tutors for a wide spectrum of computer science domains, ranging from basic programming skills such as evaluating arithmetic and Boolean expressions (Krishna & Kumar, 2001) to solving problems in automata theory (Cavalcante et al., 2004), and graph theory (Bridgeman et al., 2000). An extensive overview of tutors categorized based on: 1) the nature of the problem-solving activity they

promote; 2) the way they provide problem examples; 3) the way they provide answers; 4) grading criteria; 5) feedback generation; and 6) feedback explanation is provided in (Kumar, 2005).

TIARA Tutor Domain

TIARA tutor covers over fifty examples for the following four categories of recursive algorithms.

- Decrease-by-Constant-Factor – This category includes problems of size n that invoke one sub-problem of size n reduced by a constant factor c, where c is a positive integer that does not depend on n. The corresponding recurrences for time efficiency analysis are of type T(n) = T(n / c) + f(n). Function f(n) denotes the cost of the non-recursive portion of the code.
- Divide-and-Conquer – This category includes problems of size n that invoke a (where a is positive integer constant i.e. a >= 1) sub-problems of size reduced by positive integer constant factor c, where c >= 2. The recurrences are of type T(n) = a T(n / c) + f(n).
- Decrease-by-Constant – This category includes problems of size n that invoke one sub-problem of a size reduced by a positive integer constant c. The recurrences are of type T(n) = T(n - c) + f(n).
- General Decrease-and-Conquer – This category includes recursive problems of size n that invoke number a sub-problems of a size reduced by a positive integer constant c. Constants a and c are positive integers. The corresponding recurrences are of type T(n) = a T(n - c) + f(n).

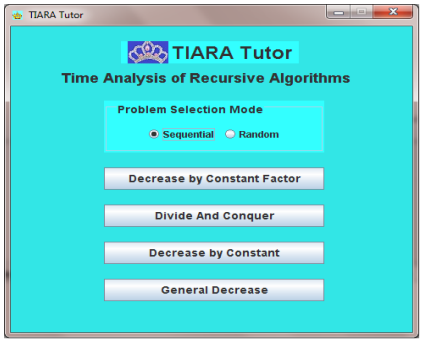
TIARA Tutor Description

There are two modes, sequential and random, of selecting the problem from the desired category list in TIARA tutor. ESRATEA was designed to always randomly select one problem from the desired category. That was sufficient for testing the learner's knowledge, but during the learning phase, sequential order is preferable because it orders the problems from simple to advanced. Random mode is more appropriate for assessing the learner's knowledge after some practicing. It displays problems randomly from the selected list until all the problems from the list are completed. After that, if learner wants to continue, the software resets.

ESRATEA had three categories: chip-and-conquer, chip-and-be-conquered and divide-and-conquer. TIARA uses more detailed categorization: decrease-by-constant-factor; divide-and-conquer; decrease-by-constant; and general decrease-and-conquer as described in *TIARA Tutor Domain* section. The names

for the category types follow the categorization applied in (Levitin, 2012). In the Figure below, it can be seen the initial screen for TIARA tutor.

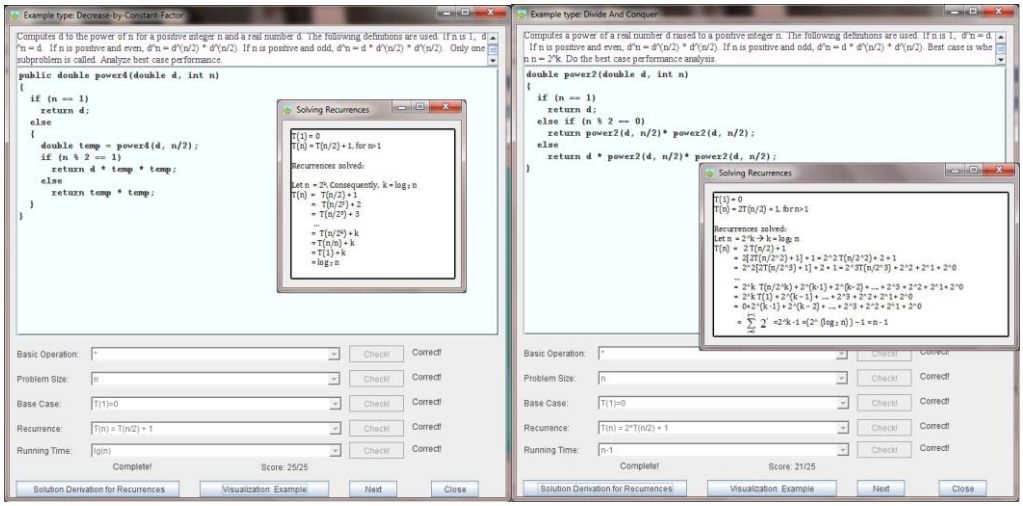
Figure 1. *Initial Screen for TIARA Tutor*



For each of the buttons there is a tool tip, which includes the recurrence relations type for the current problems. Tool tip text is displayed when the cursor is rested on top of the button.

Each problem displayed by TIARA tutor includes description in English and code for the problem implemented in Java language. Learners are asked to perform worst case time performance analysis unless specifically instructed otherwise. They have to determine the basic operation, the problem size n and recurrence relations, base case and recursive step, in this order. Each successive step can only be attempted upon successful completion of the previous steps. By the end, learners should solve the recurrences in order to calculate the resulting T(n). When the correct answer for T(n) is specified, the student may compare his/her solution derivation with the one provided by the tutor. In addition, at that time, student may also see the performance equivalent template example which shares the same recurrences as the original problem.

Figure 2. *d^n as Decrease-by-Constant* **Figure 3.** *d^n as Divide-and-Conquer*



Based on implementation, the same problem can be solved by several different algorithms. Therefore, the time analysis of different algorithms for the same problem can not only have different recurrences but they may belong to different categories. Each different algorithm implementation for the same problem displays different examples, which can be in the same category or in different categories. Figures 2 and 3 illustrate calculating power d^n with algorithms that belong to decrease-by-constant and divide-and-conquer categories, respectively.

In addition to the two examples shown above, TIARA tutor provides several other examples for calculating d^n implemented with many different algorithm types based on the following defining recurrences:

- $d^0 = 1$ and $d^n = d*d^{n-1}$, for n>0.

This algorithm has time analysis recurrences T(1)=0 and T(n)=T(n-1) +1, for n>1. We assume that multiplication is the basic operation. The algorithm is a decrease-by-constant type and has linear time.

- $d^0 = 1$ and $d^n = d*(d^{n/2})^2$, for n>0 and n odd, and $d^n = (d^{n/2})^2$, for n>0 and n even.

If only one recursive call is made to sub-problem to determine $d^{n/2}$, n is even and multiplication is the basic operation, the time analysis recurrences are T(0)=0 and T(n)=T(n/2) +1. This algorithm is a decrease-by-constant-factor and a divide-and-conquer type and has logarithmic time.

- $d^0 = 1$ and $d^n = d * d^{n/2} * d^{n/2}$, for n>0 and n odd, and $d^n = d^{n/2} * d^{n/2}$, for n>0 and n even.

If two separate recursive calls are made to call sub-problems to determine $d^{n/2}$, n is even and multiplication is the basic operation, the time analysis recurrences are T(0)=0 and T(n)=2T(n/2)+1. This algorithm is a divide-and-conquer type and has linear time.

- For special case when d=2, the defining recurrences are as follows: $2^0 = 1$, and $2^n = 2^{n-1} + 2^{n-1}$, for n>0.

If only one recursive call is made to calculate the sub-problems 2^{n-1}, and addition is the basic operation, the time analysis recurrences are T(0)=0 and T(n)=T(n-1)+1.The algorithm is a decrease-by-constant type and has linear time.

If an algorithm invokes two recursive calls to determine each 2^{n-1} separately, and addition is the basic operation, the time analysis recurrences are T(0)=0 and T(n)=2T(n-1)+1. The algorithm is a general decrease-and-conquer type and has exponential time.

Experience with Tutor

The learners provided very positive feedback after using the software as a supplemental tool in the course (Pevac, 2012). The only aspect of the tutor that was not embraced by learners was visualization examples implemented in ESRATEA (Pevac & Carpenter, 2010). The graphical examples provided were designed to draw a line of length k to illustrate that k basic operations were

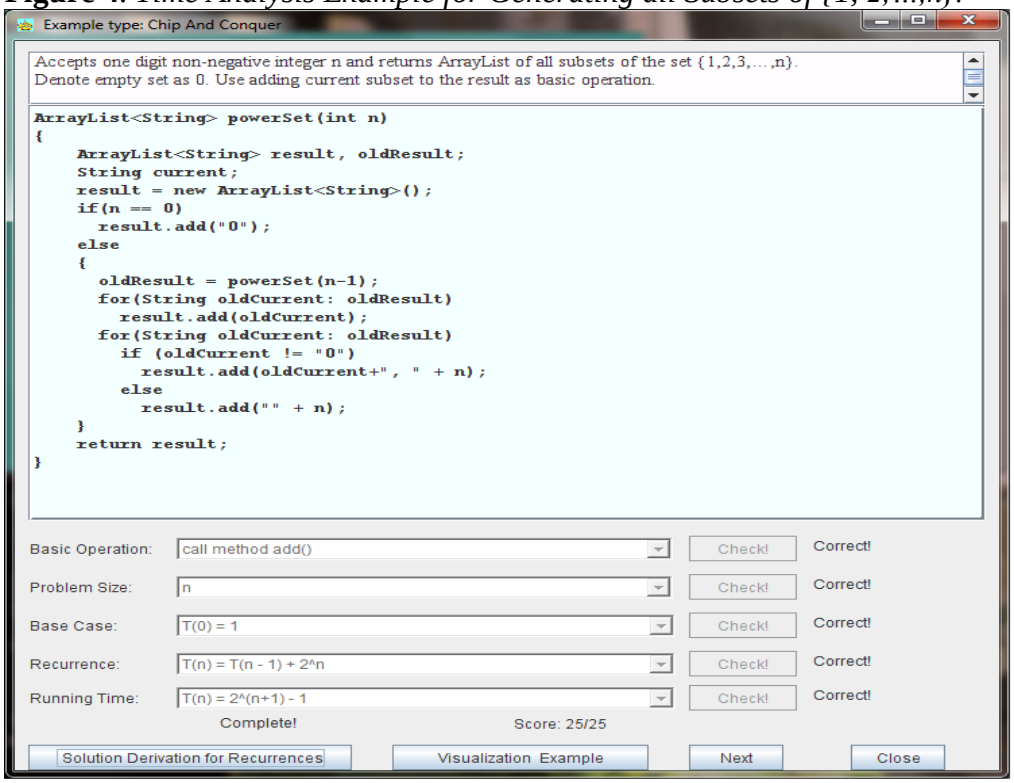
performed. The lines were drawn on a graphical surface in a way that helps to determine their total length more easily. The total length is also the solution of corresponding recurrences and represents the run time performance function T(n). Only the best students were able to grasp the analogy and recognize the equivalence classes of problems that share the same recurrences and have the same solution.

The main problem was that students who had difficulty to establish and solve the recurrences also had problem drawing lines on a two-dimensional graphical surface as discussed in (Pevac, 2012).

In order to provide visualization that would be more appropriate for a majority of learners, TIARA uses much simpler examples based on the template approach introduced in (Pevac, 2011). For each time analysis problem, the code in the visualization example prints one letter whenever the original problem performs one basic operation. All other operations in the original problem that are not basic operations are omitted. In addition, the visualization examples include plotted pictures for the run time functions T(n) and the table of values for T(n). Since the original problem and the simplified code have the same recurrences, both have the same solution for the function T(n). Consequently, both have the same asymptotic growth. The simplified visualizing examples are expected to help the learner to better recognize the relationship among the pattern of code, the recurrences type and the run time.

More Challenging Examples of Time Analysis Included in TIARA

Figure 4. *Time Analysis Example for Generating all Subsets of {1, 2,…,n}.*



In order to make TIARA tutor more appealing to students who do not need extensive practice of time analysis for simple recursive algorithms, we have included some more challenging examples of time analysis as well. The added examples are above the level of those given on tests and include time analysis of an algorithm which multiplies two matrices utilizing Strassen's method, analysis of an algorithm which multiplies two numbers via Karatsuba's method, analysis of an algorithm for generating all subsets (see Figures 4 and 5), analysis of an algorithm for generating all permutations (see Figures 6 and 7), and analysis of an algorithm to calculate the n-th Fibonacci number.

Figure 5. *Visualization and Solution Derivation for all Subsets of {1, 2,…,n}*

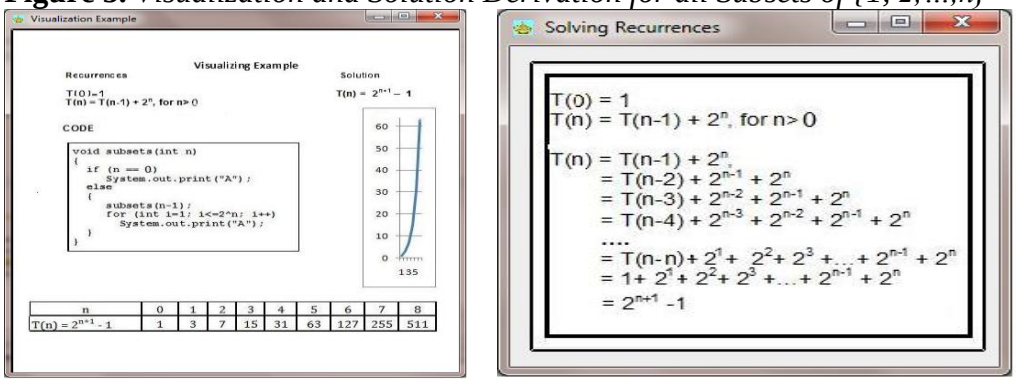


Figure 6. *Example for Generating all Permutations of 1,2,…,n.*

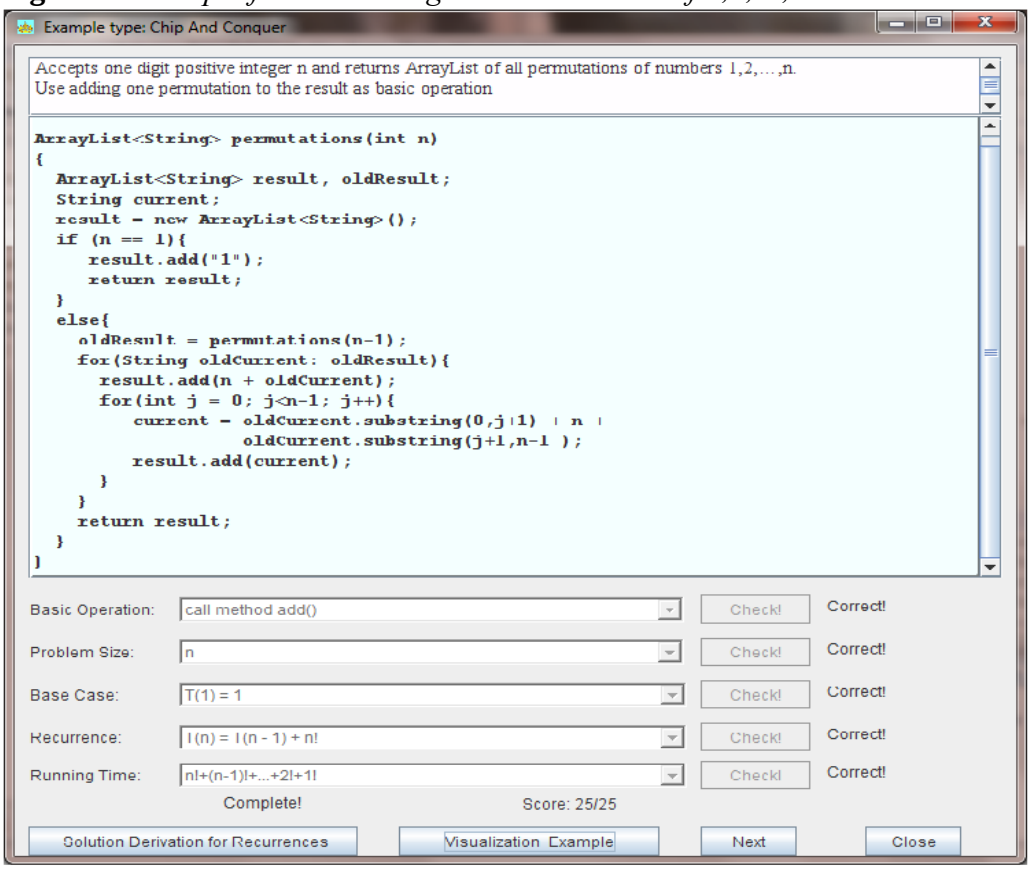
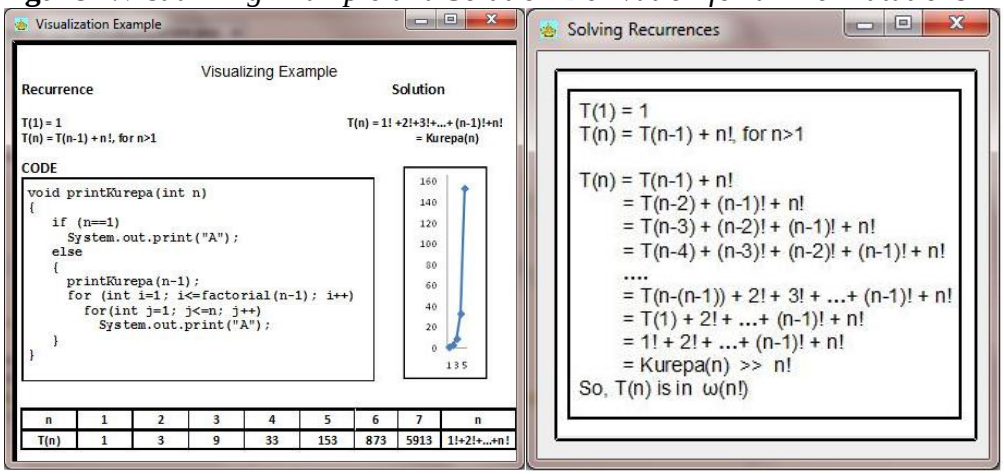


Figure 7.*Visualizing Example and Solution Derivation for all Permutations*



In Figure 7 that shows deriving the solution for the time analysis of the algorithm to generate permutations, we use Kurepa(n) to specify function 1! + 2! + .+ n!. Related function left factorial, denoted !n, was introduced by (Kurepa, 1971) to specify function 0! + 1! +…+ (n-1)!. The relationship between functions Kurepa(n) and left factorial is Kurepa(n) = !(n+1) - 1.

Conclusion

Based on the categorization of tutors described in (Krishna & Kumar, 2001), the TIARA tutor has examples carefully selected by the instructor and incorporated into the software. Correct answers were also created by the instructor and built into the software. Grading is done by the tutor in a fully automated way. Feedback is also generated by the tutor in an automated way. Feedback content displays “correct” or “incorrect” for most of the questions and provides complete derivation for solving recurrences. The visualization examples combined with practicing and requesting that learners create new problems are intended to promote knowledge, application, analysis and synthesis (according to Blooms taxonomy) for building the knowledge of time efficiency analysis of recursive algorithms.

Acknowledgments

Partial support for this work was provided by the CCSU University Research Grant. In addition, the author would like to thank to graduate students R. Kuruvela and K. Reganti, who created pictures for many simplified visualizing examples and implemented time analysis for Strassen’s multiplication and Karatsuba’s multiplication as part of their capstone project.

References

Baase, S., & Van Gelder, A. 2000, *Computer Algorithms* (3rd ed.). Addison Wesley Longman; Pearson Education. [Reprinted 2009].

Bridgeman, S., Goodrich, M. T., Kobourov, S.G., Tamassia, R. 2000, PILOT: An Interactive Tool for Learning and Grading. *Proc. of the SIGCSE Technical Symposium on Computer Science Education*, Austin, TX, ACM Press, 139-143. DOI = http://dx.doi.org/10.1145/331795.331800.

Cavalcante, R., Finley, T., Rodger, S.H. 2004. A Visual and Interactive Automata Theory Course with JFLAP 4.0. *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education)*, Vol 36 , Issue 1, 140-144. Also published in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, Norfolk, VA, March 2004, 140-144. DOI = http://doi.acm.org/10.1145/1028174.971349.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & C., Stein. 2007. *Introduction to Algorithms* (3rd ed.). MIT Press; McGraw-Hill Book Company.

Krishna A.,Kumar, A. 2001. A Problem Generator to Learn Expression Evaluation in CS I and its Effectiveness, *The Journal of Computing in Small Colleges*, Vol 16, No 4, May 2001, 34-43. DOI = http://dl.acm.org/citation.cfm? id=378659.

Kumar, A. 2005. Generation of Problems, Answers, Grade and Feedback – Case Study of a Fully Automated Tutor, *Journal of Educational Resources in Computing* (*JERIC*). Vol 5 No 3, Article no 3, Sep 2005. DOI = http://dx.doi.org/10.1145/1163405.1163408

Kurepa, Dj. 1971. On the left factorial function n*!*, *Math. Balkan*. 1, 147-153.

Levitin, A. 2012. *Introduction to the Design & Analysis of Algorithms* (3rd ed.). Pearson.

McConnell, J. J. 2008. *Analysis of Algorithms* (2nd ed.). Jones and Bartlett Publishers.

Neapolitan, R. 2014. *Foundations of Algorithms* (5thed.). Jones & Bartlett Learning.

Pevac I. Using Templates to Introduce Time Efficiency Analysis in an Algorithms Course. 2011. *WORLDCOMP'11 – FECS, Proc. of the 2011 Int. Conf. on Frontiers in Education: Computer Science & Computer Engineering*, (Ed. Arabnia, Clincy, Deligiannidis), Las Vegas, NV, SCREA Press, 373-379. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.217.8911

Pevac, I. First Experiences with Software Tool for Recursive Algorithm Time Efficiency Analysis. 2012. *The Journal of Computing Sciences in Colleges*. Vol 28, Num 1, Oct 2012, 56-65. http://dl.acm.org/citation.cfm?id=2379714

Pevac, I., Carpenter T. 2010. ESRATEA Software for Recursive Algorithms Time Efficiency Analysis. WORLDCOMP'10 – FECS, *Proc. of the 2010 Int. Conf. on Frontiers in Education: Computer Science & Computer Engineering*, (Hamid R. Arabnia, Victor A. Clincy, Azita Bahrami, Ashu M. G. Solo Eds), Las Vegas, NV, July 2010, CSREA Press, 367-373. http://www.informatik.uni-trier.de/~ley/db/conf/fecs/fecs2010.html

Pothering, G., Naps, T. 2002. *Introduction to Data Structures and Algorithms Analysis with C++*, West Publishing Company.