European Journal of Technique

journal homepage: https://dergipark.org.tr/en/pub/ejt

Vol.13, No.1, 2023

Enhancing Algorithm Design Education: Interactive Learning with Jupyter Notebooks

Oguzhan Topsakal[1*]

[1*]Florida Polytechnic University, 33805, Lakeland, Florida, USA. (e-mail: otopsakal@floridapoly.edu).

ARTICLE INFO

Received: Jan., 04. 2023
Revised: Jun., 27. 2023
Accepted: Jun, 30. 2023

Keywords:
Algorithm
Design
Approach
Technique
Jupyter Notebook
Interactive
Teaching
Education
Learning

Corresponding author: *Oguzhan Topsakal*

ISSN: *2536-5010* | e-ISSN: *2536-5134*

DOI: https://doi.org/10.36222/ejt.1320404

ABSTRACT

Interactive learning environments play a crucial role in facilitating effective education. Traditional approaches to teaching algorithm design techniques often lack interactivity, resulting in a limited learning experience. In this article, we propose the utilization of Jupyter Notebooks, renowned for their versatility in combining code, visualizations, and explanations, as a powerful tool for enhancing understanding and promoting an engaging learning experience in teaching algorithm design techniques. We provide a comprehensive structure for a Jupyter Notebook page, encompassing problem descriptions, brute force solutions, algorithm design techniques, application areas, and references, to present a thorough solution for computer science problems using these design techniques. Our study includes experiments conducted with computer science students, demonstrating the practical application of Jupyter Notebooks in algorithm design education. Furthermore, we share sample Jupyter Notebooks for popular algorithm design techniques, including Divide & Conquer, Greedy, Dynamic Programming, and Backtracking. Importantly, we emphasize the significance of interactive comparisons between brute force solutions and algorithm design techniques, which foster valuable learning opportunities by providing insights into performance improvements, complexity analysis, validation, optimization strategies, trade-offs, and a deeper understanding of algorithmic principles. In conclusion, we propose the integration of Jupyter Notebooks as a potent tool for teaching algorithm design approaches, empowering learners to actively engage with the material, visualize complex concepts, and collaborate effectively. By incorporating Jupyter Notebooks into algorithm design education, instructors can enhance students' comprehension, cultivate critical thinking skills, and facilitate a profound understanding of algorithmic principles and optimization strategies.

1. INTRODUCTION

The famous quote, "Tell me and I forget, teach me and I may remember, involve me and I learn," underscores the pivotal role of interaction in effective learning. Algorithm design, being an abstract concept, is best taught through active involvement. Algorithm design approaches offer techniques for solving computer science problems, resulting in improved performance in terms of time complexity. [1,2] However, the conventional teaching methods employed in classrooms and textbooks primarily revolve around examples of computer science problems with limited or no interactive elements. [3] Consequently, much of the learning diminishes over time.

To address this issue, we propose the utilization of Jupyter Notebooks—an open-source web-based interactive computing environment—as a valuable tool for enhancing understanding and facilitating effective teaching of algorithm design approaches and time complexity analysis. [4] Jupyter Notebooks provide users with a flexible platform for creating and sharing interactive documents called notebooks. They support multiple programming languages, including Python, making them versatile for various data analysis and scientific computing tasks.

By leveraging the interactive and exploratory workflow offered by Jupyter Notebooks, learners can actively engage in running code snippets, visualizing results, and iteratively modifying their analyses. Jupyter Notebooks empower users to seamlessly combine code, visualizations, and explanatory text within a single document, fostering an interactive and comprehensive learning experience.

In this study, we have developed Jupyter Notebook pages that incorporate detailed descriptions, code implementations, and visualizations to present practical examples of algorithm design approaches (techniques). These notebooks serve as a powerful methodology for teaching algorithm design, enabling learners to actively participate in interactive sessions and gain a deep understanding of the approaches employed. We strongly believe that the adoption of Jupyter Notebooks as a standard practice for teaching algorithm design will greatly enhance the educational experience and contribute to improved learning outcomes.

This article advocates for the use of Jupyter Notebooks as an invaluable tool to promote understanding of algorithm design approaches and time complexity analysis through interactive learning sessions. By offering a platform that integrates code, visualizations, and explanatory text, Jupyter Notebooks empower learners to delve into algorithmic problem-solving, fostering a deeper comprehension of the subject matter. It is our hope that the methodology presented here, employing Jupyter Notebooks for teaching algorithm design, becomes widely embraced to enhance the teaching and learning experience in algorithm design education.

2. RELATED RESEARCH

Efforts have been made to enhance algorithm teaching through the utilization of analogies and metaphors [5,6]. For instance, the book "Explaining Algorithms Using Metaphors" aims to equip teachers with metaphors and analogies to facilitate the comprehension of algorithmic concepts by students. [6] This comprehensive resource offers guidance on various subjects, including Graph Algorithms, Computational Geometry, and Strings and Sequences. Each subject is accompanied by background information, metaphorical explanations, analytical insights, teaching experiences, and relevant exercises.

In addition to algorithm teaching, Jupyter Notebooks have emerged as valuable collaboration and teaching tools across multiple disciplines, such as architecture [7], mathematics [8], and geography [9]. Castelo-Branco et al. proposed the utilization of Jupyter Notebooks in architecture to foster collaborative work and seamlessly integrate text, formulas, data, code, and graphics, thereby preserving the design narrative and supporting reproducibility [7]. Ketcheson highlights the benefits of Jupyter Notebooks in a Masters-level numerical analysis course, sharing detailed examples based on personal teaching experiences [8]. Similarly, Reades leveraged Jupyter Notebooks to teach geocomputation modules to Geography undergraduates [9].

Numerous educators have shared their best practices for utilizing Jupyter Notebooks in teaching [9-11]. Johnson critically examines the strengths and weaknesses of Jupyter notebooks for education, drawing on extensive experience teaching various courses, and provides a set of recommended best practices [10]. Leitão and Teixeira propose Jupyter Notebooks as a tool to foster the development of innovative teaching methodologies [11].

In the field of computer science, Jupyter Notebooks have also gained recognition and endorsement. For example, Liubschenko and Parkomenko explore the pedagogical and technical challenges of using notebooks as an educational tool, evaluating their effectiveness based on educational criteria such as complexity, interactivity, and utility [12]. Glick and Mache employ Jupyter Notebooks to create an open-access course on high-performance computing, reporting improved interactivity and enhanced outcomes [13]. Hamouda et al. develop interactive Jupyter Notebooks as a tool for teaching recursion, resulting in improved performance in recursion exam questions compared to students using traditional instruction methods [14]. Birster adopts individual Jupyter Notebook pages at different levels of Bloom's Taxonomy to enhance engagement among computer science students [15].

The widespread adoption and endorsement of Jupyter Notebooks across various disciplines, including computer science, reinforce its effectiveness as a versatile tool for teaching and collaboration. These experiences and best practices contribute to the growing body of evidence supporting the use of Jupyter Notebooks in educational contexts.

3. METHODOLOGY

This section outlines the methodology employed in teaching algorithm design approaches using Jupyter Notebooks. The algorithm design approaches are categorized into four main categories: divide & conquer, greedy, backtracking, and dynamic programming. A general description of each design approach and the corresponding steps involved in applying the approach is provided.

To exemplify the application of these design approaches, multiple Jupyter Notebooks were prepared, each focusing on a specific computer science problem such as traveling salesman, tug-of-war, longest common string, and closest pair of points. Each Jupyter Notebook page comprises several sections, including:

1. Introduction of the Computer Science (CS) Problem:
    - A comprehensive description of the problem,
    - Sample input and output cases,
    - Visuals, such as images or short videos, to depict the problem visually,
2. Brute Force Approach to Solve the CS Problem:
    - Description of the brute force solution
    - Visualization of the asymptotic complexity, showcasing the time required for executing the brute force algorithm as a function of the input size,
    - Graphical representations, such as Figure 1, demonstrating the time complexity of the brute force solution, e.g., O(n^2) for the 'Closest Pair of Points' problem,
3. Applying one of the Algorithm Design Approaches:
    - Step-by-step instructions to solve the computer science problem using the chosen algorithm design technique,
    - Visuals elucidating the algorithm design technique employed,
    - Pseudocode representing the solution using the design technique in plain English,
    - Python implementation code for applying the approach, including detailed comments for better comprehension,
    - Driver code with sample inputs, enabling the presentation of results on the Jupyter Notebook page,
    - Asymptotic complexity analysis of the solution, highlighting the time and space complexity optimizations achieved,

4. Application Areas of the CS Problem:
   - Identification and enumeration of the practical application areas associated with the computer science problem under consideration,
   - For instance, the 'closest pair of points' problem can find application in determining facility location optimization, enabling the selection of optimal locations for new facilities or services based on proximity to existing locations or customer demand,

Additionally, references to external resources are provided for further exploration and understanding of the computer science problem and the algorithmic approach utilized.

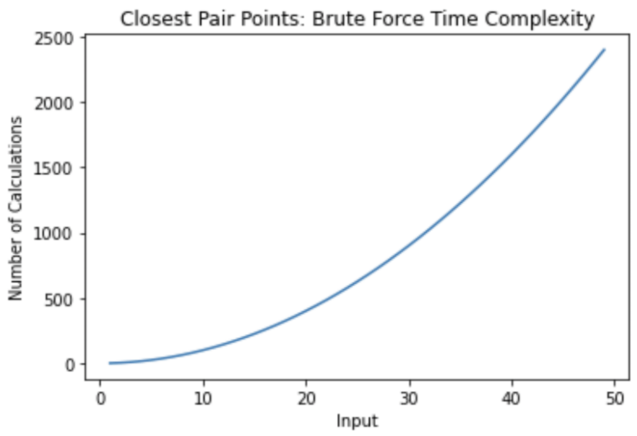

Figure 1. The asymptotic time complexity of brute force solution for the closest pair of points computer science problem.

The steps involved in each algorithm design approach category are summarized in Table 1. However, the steps might need to be tailored to the specific computer science problem being discussed. By offering detailed explanations, pseudocode, and Python implementation code, learners gain insights into the optimization strategies and techniques used to develop efficient algorithms. Moreover, the Jupyter Notebooks facilitate the visual representation of the problem, enabling learners to grasp the input-to-output transformation. The inclusion of the brute force solution as a baseline allows for comparative analysis, showcasing the improvements achieved through the algorithm design techniques.

The Python implementation of the algorithm for the given computer science problem is provided, utilizing the designated algorithm design technique. The implementation includes comprehensive code comments to enhance understanding and can be presented in one or multiple code cells within the Jupyter Notebook. Subsequently, a separate code cell encompasses the driver code, incorporating sample inputs. Upon running the driver code, the Jupyter Notebook page displays the resulting outputs. To analyze the algorithm's performance, the driver code can be executed with inputs representing increasing input sizes. The execution times for each input size can be recorded and plotted on a graph.

For example, Figure 2 illustrates the theoretical and experienced execution times for the Closest Pair of Points problem. The theoretical execution time (O(nlogn)) based on the input size, is depicted under the 'Ideal Graph' section at the bottom, and the experienced execution time is represented at the top of Figure 2. Similarly, Figure 3 presents the theoretical and experienced execution times for the Box Stacking problem.

This proposed structure of a Jupyter Notebook strives to present a comprehensive and interactive learning experience, allowing learners to understand algorithm design approaches and their practical application.

TABLE I

MAIN STEPS OF ALGORITHM DESIGN APPROACHES

Design Category	**Steps**
Divide & Conquer	**Divide**: Break down the problem into smaller, more manageable subproblems. **Conquer**: Solve the subproblems recursively or by applying a base case. **Combine**: Merge the solutions of the subproblems to obtain the final solution to the original problem.
Greedy Approach	**Initialization**: Set up the initial solution or state. **Determine the Greedy Choice**: Make the locally optimal choice at each step, assuming it will lead to a globally optimal solution. **Feasibility Check**: Verify if the greedy choice is valid. **Update**: Update the current solution with the chosen greedy choice. **Termination**: Determine when to stop and return the final solution.
Backtracking	**Choose**: Select a candidate for the current step or decision. **Constraints Check**: Verify if the chosen candidate satisfies the problem constraints. **Apply**: Apply the chosen candidate to the current step and move to the next step. **Recurse**: Recursively call the algorithm on the next step. **Backtrack**: Undo the previous step and go back to explore other candidates if the current candidate does not lead to a valid solution. **Termination**: Determine when to stop and return the final solution.
Dynamic Programming	**Determine the objective**: Clearly define the problem and determine the objective and constraints. **Identify Overlapping Subproblems**: Identify the subproblems that overlap or are repeated in the problem-solving process. **Formulate Recurrence Relation**: Define a recurrence relation that expresses the solution to a larger problem in terms of the solutions to its subproblems. **Create a Memorization Table or Array**: Set up a data structure to store the solutions to subproblems to avoid redundant computations. **Solve Bottom-Up or Top-Down**: Choose either a bottom-up or top-down approach to iteratively or recursively solve the problem by building on the solutions to subproblems. **Build the Final Solution**: Construct the final solution by combining the solutions to subproblems according to the recurrence relation.

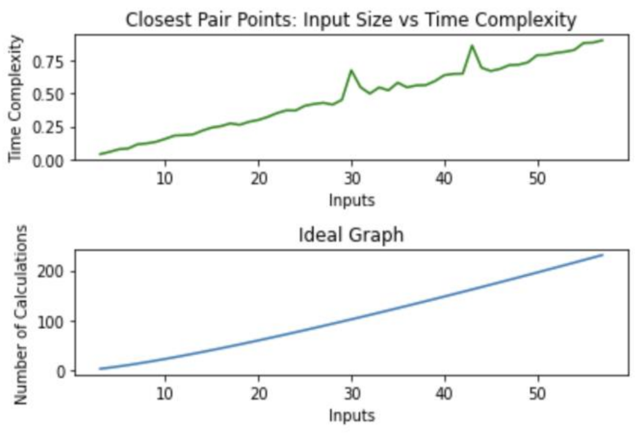

Figure 2. The theoretical and experienced execution times for the Closest Pair of Points problem.

Overall, this methodology empowers learners to actively engage with the material, experiment with different approaches, and acquire a deeper understanding of algorithmic principles and optimization strategies.

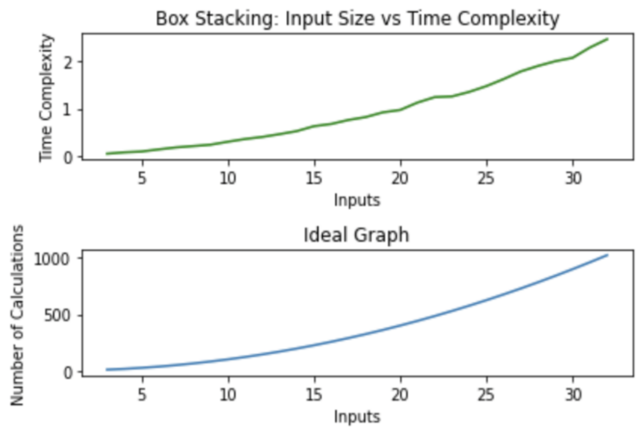

Figure 2. The theoretical and experienced execution times for the Box Stacking problem are presented on a Jupyter Notebook page.

4. EXPERIMENTS

In the algorithm design and analysis course, students specializing in Computer Science were instructed to select a computer science problem of their choice and employ one of the algorithm design techniques to present its solutions. To facilitate their presentations, the aforementioned sections were provided as a structured template, guiding the creation of Jupyter notebook pages. Some of the students granted permission to publish their refined Jupyter Notebook pages, which are now available on a dedicated website[1] [16].

Moreover, the website serves as a comprehensive resource, organizing well-known computer science problems into various algorithm design techniques. Each problem is accompanied by a concise description. The website's extensive list comprises a total of 118 computer science problems, with 27 categorized under dynamic programming, 41 under backtracking, 30 under divide and conquer, and 24 under greedy algorithms. This website aims to provide a valuable platform for showcasing the Jupyter Notebooks as a practical educational tool for algorithm design techniques in solving diverse computer science problems and to contribute to the academic community by fostering collaboration and sharing algorithmic insights.

5. DISCUSSION

Utilizing Jupyter Notebooks for teaching algorithm design approaches can significantly enhance understanding and facilitate the development and exploration of algorithms. Jupyter Notebooks provides an interactive environment where learners can write, execute, and modify code snippets in real-time. This allows the student to experiment with different algorithms, tweak parameters, and observe the immediate results. The interactive nature of Jupyter Notebooks promotes an iterative and exploratory approach to algorithm design. Jupyter Notebooks allow you to include rich text, markdown cells, and visualizations alongside your code. This helps provide detailed explanations, document the thought process, describe the algorithm steps, and include mathematical equations or diagrams to clarify concepts. The ability to combine code and explanations in a single document enhances understanding and promotes self-contained documentation. Moreover, Jupyter Notebooks support the integration of various data visualization libraries, such as Matplotlib or Seaborn. This enables you to visualize data, plot graphs, or display intermediate results during algorithm execution. Visualizations help you grasp patterns, observe algorithm behavior, and gain insights into the underlying data structures or problem-solving techniques. The implementation can be spread into various code cells, which can execute code cells step-by-step, observing the intermediate results at each stage. This is particularly useful for complex algorithms or iterative processes, allowing you to examine the algorithm's progress, verify intermediate computations, and identify any potential issues or bugs. Another great benefit of Jupyter Notebooks is enabling users to share them easily through cloud-based servers such as Google Colab, allowing for collaboration and knowledge exchange among colleagues, researchers, or instructors. The ability to collaborate and receive input from others can enhance your understanding and help refine your algorithms. Jupyter Notebooks also promote reproducibility by enabling others to reproduce your algorithms, verify your findings, or build upon your work.

The Jupyter Notebooks make it possible to review the theoretical and practical outputs of asymptotic complexity results side by side, as seen in Figure 2, and help to achieve a better understanding of the concepts. To better understand the asymptotic complexity, Jupyter Notebooks offer an interactive, visual, and collaborative learning experience. By combining code execution, visualizations, mathematical equations, explanations, and documentation in a single document, Jupyter Notebooks enhance your comprehension of complexity analysis concepts and provide a flexible platform for exploration, analysis, and sharing.

The proposed content for teaching algorithm design approaches via interactive Jupyter Notebooks encompasses sections to compare the brute force solution with the solution achieved by applying an algorithm design technique. Comparing the brute force solution with the solution achieved by applying an algorithm design technique provides insights into performance improvements, complexity analysis, validation, optimization strategies, trade-offs, and learning opportunities. It enables you to evaluate and appreciate the benefits of algorithm design techniques in solving computational problems efficiently and effectively. These benefits can be explained as follows:

- *Performance Evaluation*: By comparing the two solutions, you can assess the performance improvement achieved through the algorithm design technique. This includes evaluating factors such as execution time, memory usage, and overall efficiency. Comparing the performance metrics allows you to quantify the gains achieved by employing the algorithm design technique and determine its effectiveness in optimizing the solution.

[1] Interactive Examples of Algorithm Design Techniques: https://sites.google.com/view/algorithm-design-approaches/

- *Complexity Analysis*: Comparing the brute force solution with the algorithm design technique provides an opportunity to analyze the time and space complexity of both approaches. It helps you understand how the algorithm design technique reduces the computational requirements compared to the brute force solution. This analysis enhances your understanding of algorithmic efficiency and scalability based on the input size.
- *Validation and Verification*: Comparing the solutions helps validate the correctness of the algorithm design technique. By verifying that the results obtained from both approaches are equivalent, you can gain confidence in the correctness of the optimized solution. This verification process is crucial to ensure that the algorithm design technique produces accurate and reliable results and provides the expected efficiency.
- *Insight into Optimization*: Analyzing the differences between the brute force solution and the optimized solution offers insights into the optimization strategies employed by the algorithm design technique. You can identify the key optimizations or algorithmic improvements that lead to enhanced performance. Understanding these optimizations helps you grasp the underlying principles and techniques used to refine algorithms and improve efficiency.
- *Trade-offs and Constraints*: Comparing the two solutions allows you to explore trade-offs between various factors, such as time complexity, space complexity, accuracy, or implementation complexity. The brute force solution might provide a baseline that guarantees correctness but might not be feasible for larger problem instances due to its inefficiency. The optimized solution achieved through the algorithm design technique can offer a balance by addressing these constraints and providing a more practical solution.
- *Learning Opportunity*: The comparison between the brute force solution and the optimized solution can serve as a valuable learning experience. It helps you understand the importance of algorithmic efficiency, explore different algorithm design techniques, and appreciate the impact of optimizations on solving complex problems. By analyzing the differences and studying the optimized solution, you can enhance your algorithmic thinking and problem-solving skills.

6. CONCLUSION

In conclusion, the use of Jupyter Notebooks as a platform for teaching algorithm design is a valuable tool for enhancing understanding and promoting effective learning experiences. By combining code execution, visualizations, and explanations in a single document, Jupyter Notebooks offer an interactive and versatile environment for exploring algorithms and analyzing their asymptotic complexity.

We demonstrated how Jupyter Notebooks can be utilized to teach algorithm design approaches effectively. By categorizing the approaches into distinct categories and providing examples of Jupyter Notebook pages, learners are guided through the process of understanding and implementing different algorithmic techniques. The structure of the Jupyter Notebook pages, including problem description, brute force solutions, algorithm design techniques, application areas, and references, ensures a comprehensive learning experience.

The experiments conducted with computer science students further validate the efficacy of using Jupyter Notebooks for algorithm design education. The students' engagement in creating and presenting their own algorithmic solutions using Jupyter Notebooks showcases the benefits of interactive exploration, visualization, step-by-step execution, and collaboration. The integration of theoretical and practical outputs of asymptotic complexity analysis within the notebooks enhances learners' understanding of algorithmic concepts.

We also emphasize the importance of comparing brute-force solutions with algorithm design techniques. Such comparisons provide insights into performance improvements, complexity analysis, optimization strategies, and trade-offs. Learners gain a deeper understanding of algorithmic efficiency, scalability, and problem-solving techniques through this comparative analysis.

Overall, this article advocates for the broader adoption of Jupyter Notebooks in teaching algorithm design approaches. The interactive and collaborative nature of Jupyter Notebooks, along with their ability to integrate code, visualizations, explanations, and documentation, significantly enhance the comprehension and application of algorithmic concepts. By embracing Jupyter Notebooks as a pedagogical tool, educators can create a more engaging and effective learning environment, enabling learners to actively participate, explore, and master algorithm design approaches.

ACKNOWLEDGEMENT

We would like to thank Taylor Preston and Joshua Murphy for their help in editing selected Jupyter Notebooks and also thank Florida Polytechnic University Computer Science students who let their Jupyter Notebooks be utilized on this study’s website.

REFERENCES

[1] Neapolitan, R., & Naimipour, K. (2010). Foundations of algorithms. Jones & Bartlett Publishers.

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms. MIT press.

[3] Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2016). Expanding computer science education in schools: understanding teacher experiences and challenges. Computer science education, 26(4), 235-254.

[4] Project Jupyter, https://jupyter.org/ (accessed Jun. 17, 2023).

[5] Forišek, M., Steinová, M. (2013). Explaining Algorithms Using Metaphors. Springer Briefs in Computer Science. Springer, London. https://doi.org/10.1007/978-1-4471-5019-0_1.

[6] Forišek, M., & Steinová, M. (2012, February). Metaphors and analogies for teaching algorithms. In Proceedings of the 43rd ACM technical symposium on Computer Science Education (pp. 15-20).

[7] Castelo-Branco, R., Caetano, I., Pereira, I., & Leitão, A. (2020, November). The collaborative algorithmic design notebook. In Int. Conf. of the Architectural Science Association (pp. 1056-1065).

[8] Ketcheson, D. I. (2014). Teaching numerical methods with IPython notebooks and inquiry-based learning. Proceedings of the 13th Python in Science Conference. Retrieved from http://hdl.handle.net/10754/346689.

[9] Reades, J. (2020). Teaching on jupyter. Region, The Journal of Ersa, 7(1), 21-34.

[10] Johnson, J. W. (2020, October). Benefits and pitfalls of jupyter notebooks in the classroom. In Proceedings of the 21st Annual Conference on Information Technology Education (pp. 32-37).

[11] Cardoso, A., Leitão, J., & Teixeira, C. (2019). Using the Jupyter notebook as a tool to support the teaching and learning processes in engineering courses. In The Challenges of the Digital Transformation in Education: Proceedings of the 21st International Conference on Interactive Collaborative Learning (ICL2018)-Volume 2 (pp. 227-236). Springer International Publishing.

[12] V. Liubchenko and H. Parkhomenko, "The Involvement of Jupyter Notebooks as an Educational Tools: A Case Study," 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT), LVIV, Ukraine, 2021, pp. 147-150, doi: 10.1109/CSIT52700.2021.9648674.

[13] Glick, B., & Mache, J. (2018). Using Jupyter notebooks to learn high-performance computing. Journal of Computing Sciences in Colleges, 34(1), 180-188.

[14] Hamouda, S., Edwards, S. H., Elmongui, H. G., Ernst, J. V., & Shaffer, C. A. (2018). RecurTutor: An interactive tutorial for learning recursion. ACM Transactions on Computing Education (TOCE), 19(1), 1-25.

[15] Birster, C. A. (2017). Engagement in Foundational Computer Science Courses Through Supplementary Content for Algorithms. arXiv preprint arXiv:1801.06047.

[16] Applications of Algorithm Design Approaches to Computer Science Problems Presented via Interactive Jupyter Notebook Pages, https://sites.google.com/view/algorithm-design-approaches/ (accessed June 17, 2023)

BIOGRAPHY

Oguzhan Topsakal received his B.S. in Computer Engineering in 1996 from Istanbul Technical University, Turkey. He received his M.S. and Ph.D. in Computer Science from the University of Florida in 2003 and 2007. After gaining extensive experience in the software industry, Dr. Topsakal is currently an assistant professor in the Computer Science department at the Florida Polytechnic University since 2018. He teaches courses related to Machine Learning, Algorithm Design, Databases, and Mobile Development. Dr. Topsakal's research interests include algorithm design and applications of machine learning and deep learning.