Algorithmic Design and Techniques at edX: Syllabus

June 16, 2019

Contents

1 Welcome!

Thank you for joining Algorithmic Design and Techniques at edX! In this course, you will learn not only the basic algorithmic techniques, but also how to write efficient, clean, and reliable code.

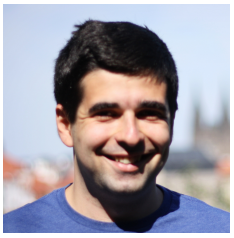
2 Who This Class Is For

Programmers with basic experience looking to understand the practical and conceptual underpinnings of algorithms, with the goal of becoming more effective software engineers. Computer science students and researchers as well as interdisciplinary students (studying electrical engineering, mathematics, bioinformatics, etc.) aiming to get more profound understanding of algorithms and hands-on experience implementing them and applying for real-world problems. Applicants who want to prepare for an interview in a high-tech company.

3 Meet Your Instructors

Daniel Kane is an associate professor at the University of California, San Diego with a joint appointment between the Department of Computer Science and Engineering and the Department of Mathematics. He has diverse interests in mathematics and theoretical computer science, though most of his work fits into the broad categories of number theory, complexity theory, or combinatorics.

Alexander S. Kulikov is a senior research fellow at Steklov Mathematical Institute of the Russian Academy of Sciences, Saint Petersburg, Russia and a lecturer at the Department of Computer Science and Engineering at University of California, San Diego, USA. He also directs the Computer Science Center in Saint Petersburg that provides free advanced computer science courses complementing the standard university curricula. Alexander holds a Ph. D. from Steklov Mathematical Institute. His research interests include algorithms and complexity theory. He co-authored online courses "Data Structures and Algorithms" and "Introduction to Discrete Mathematics for Computer Science".



Michael Levin is an Associate Professor at the Computer Science Department of Higher School of Economics, Moscow, Russia and the Chief Data Scientist at the Yandex.Market, Moscow, Russia. He also teaches Algorithms and Data Structures at the Yandex School of Data Analysis.

Pavel Pevzner is Ronald R. Taylor Professor of Computer Science at the University of California, San Diego. He holds a Ph. D. from Moscow Institute of Physics and Technology, Russia and an Honorary Degree from Simon Fraser University. He is a Howard Hughes Medical Institute Professor (2006), an Association for Computing Machinery Fellow (2010), an International Society for Computational Biology Fellow (2012), and a Member of the the Academia Europaea (2016). He has authored the textbooks Computational Molecular Biology: An Algorithmic Approach (2000), An Introduction to Bioinformatics Algorithms (2004) (jointly with Neil Jones), and Bioinformatics Algorithms: An Active Learning Approach (2014) (jointly with Phillip Compeau). He co-authored online courses "Data Structures and Algorithms", "Bioinformatics", and "Analyze Your Genome!" that are available at Coursera and edX.

Neil Rhodes is a lecturer in the Computer Science and Engineering department at the University of California, San Diego and formerly a staff software engineer at Google. Neil holds a B.A. and M.S. in Computer Science from UCSD. He left the Ph.D. program at UCSD to start a company, Palomar Software, and spent fifteen years writing software, books on software development, and designing and teaching programming courses for Apple and Palm. He's taught Algorithms, Machine Learning, Operating Systems, Discrete Mathematics, Automata and Computability Theory, and Software Engineering at UCSD and Harvey Mudd College in Claremont, California.

4 Prerequisites

Basic knowledge of at least one programming language (`C`, `C++`, `Java`, `JavaScript`, `Python2`, `Python3`, `Scala`): loops, arrays, stacks, recursion. Basic knowledge of mathematics: proof by induction, proof by contradiction.

5 Course Overview

This course is a mix of theory and practice: you will learn algorithmic techniques for solving various computational problems and will implement about 30 algorithmic coding problems in a programming language of your choice. No other online course in Algorithms even comes close to offering you a wealth of programming challenges that you may face at your next job interview. To prepare you, we invested over 3000 hours into designing our challenges as an alternative to multiple choice questions that you usually find in MOOCs. Sorry, we do not believe in multiple choice questions when it comes to learning algorithms. . . or anything else in computer science! For each algorithm you develop and implement, we designed multiple tests to check its correctness and running time—you will have to debug your programs without even knowing what these tests are! It may sound difficult, but we believe it is the only way to truly understand how the algorithms work and to master the art of programming.

Course Outline

Week 1: Programming Challenges. We will provide an overview of where algorithms and data structures are used (hint: everywhere) and walk you through a few sample programming challenges. The programming challenges represent an important (and often the most difficult!) part of this course because the only way to fully understand an algorithm is to implement it. Writing correct and efficient programs is hard; please don't be surprised if they don't work as you planned—our first programs did not work either! We will help you on your journey through the course by showing how to implement your first programming challenges. We will also introduce testing techniques that will help increase your chances of passing assignments on your first attempt. In case your program does not work as intended, we will show how to fix it, even if you don't yet know which test your implementation is failing on.

Week 2: Algorithmic Warm-up. You will learn that programs based on efficient algorithms can be a billion times faster than programs based on naive algorithms. You will learn how to estimate the running time and memory of an algorithm without ever implementing it. Armed with this knowledge, you will be able to compare various algorithms, select the most efficient ones, and finally implement them to solve various programming challenges!

Week 3: Greedy Algorithms. You will learn about seemingly naive yet powerful greedy algorithms. After learning the key idea behind the greedy algorithms, some of our students feel that they represent the algorithmic Swiss army knife that can be applied to solve nearly all programming challenges in this book. Be warned: since this intuitive idea rarely works in practice, you have to prove that your greedy algorithm produces an optimal solution!

Week 4: Divide-and-Conquer. You will learn about divide-and-conquer algorithms that will help you to search huge databases a million times faster than brute- force algorithms. Armed with this algorithmic technique, you will learn that the standard way to multiply numbers (that you learned in the grade school) is far from being the fastest! We will then apply the divide-and-conquer technique to design fast sorting algorithms. You will learn that these algorithms are optimal, i.e., even the legendary computer scientist Alan Turing would not be able to design a faster sorting algorithm!

Weeks 5-6: Dynamic Programming. You will implement various dynamic programming algorithms and will see how they solve problems that evaded all attempts to solve them using greedy or divide-and-conquer strategies. There are countless applications of dynamic programming in practice ranging from searching for similar Internet pages to gene prediction in DNA sequences. You will learn how the same idea helps to automatically make spelling corrections and to find the differences between two versions of the same text.

6 Learning Objectives

You will be able to apply the right algorithms and data structures in your day-to-day work and write programs that work in some cases many orders of magnitude faster. You'll be able to solve algorithmic problems like those used in the technical interviews at Google, Facebook, Microsoft, Yandex, etc. If you do data science, you'll be able to significantly increase the speed of some of your experiments.

7 Estimated Workload

We expect this course will take you 8–10 hours per week to complete.

8 Grading

Your grade will be composed out of the following components: six programming assignments (PA's) and the final proctored exam.

assignment	weight	# problems	# droppable
PA1: Programming Challenges	5%	2	0
PA2: Algorithmic Warm-up	12%	7	3
PA3: Greedy Algorithms	12%	6	3
PA4: Divide-and-Conquer	12%	6	3
PA5: Dynamic Programming 1	12%	5	2
PA6: Dynamic Programming 2	12%	3	1
Final exam	35%	4	1

The passing thresholds are $\geq 70\%$ for C, $\geq 80\%$ for B, and $\geq 90\%$ for A. To receive a certificate, you need to get at least C. Passing Grade: you must score 70% or above to pass the course. To get a university course credit, you must score at least 80%.

9 Deadlines

This course is self-paced, so there are no deadlines for any specific assignment. This course will be open until April 1, 2020. Shortly after that we expect to re-open the course, self-paced, with updates.

10 Verified Certificate

There are a number of differences for verified learners compared with those just wishing to audit the course.

- **Programming challenges.** Verified learners submit a source code that is then run by the autograder on a set of carefully prepared test cases. Therefore, to solve a programming challenge, a verified learner needs to implement a reliable (that works correctly on all test cases) and efficient (that works in less than one second even on massive datasets) program. Writing fast and correct programs is an important skill of a software engineer. On the contrary, audit learners instead solve a programming quiz where the goal is to submit a result for a single dataset. Hence, for such quizzes, we only check correctness on a single dataset, and we don't check efficiency at all.
- **Private forum threads.** Verified learners have access to private forum threads where they can get help on programming challenges as well as discuss various issues with other verified learners who appreciate the impact of the grade on your potential certificate.
- **Official proof of completion.** By completing the course, verified learners receive an easily shareable official certificate. To earn a verified certificate, you need to be enrolled as part of the verified track, complete identity verification before you take the proctored final exam, and earn a passing grade. If you are auditing the course, you will not receive a certificate.

- **University course credit.** This class is a part of MicroMasters program "Algorithms and Data Structures". By completing the MicroMasters program, you indicate to employers and hiring personnel that you have made a significant investment in completing rigorous, Masters-level courses and content.

  Learners who successfully earn the Algorithms and Data Structures MicroMasters Credential are eligible to apply for admission to the School of Individualized Study (SOIS) Master of Science in Professional Studies at Rochester Institute of Technology.

  If a learner applies for admission to the SOIC Master of Science in Professional Studies program at Rochester Institute of Technology, and is accepted, the MicroMasters Credential will count towards 25% of the coursework required by this program.

  To receive a MicroMasters certificate, you must receive verified certificates in all eight courses in the program. This course is worth 20 credits. The other courses in the MicroMasters program:

  - Data Structures Fundamentals (15 credits)
  - Graph Algorithms (15 credits)
  - NP-Complete Problems (10 credits)
  - String Processing and Pattern Matching Algorithms (10 credits)
  - Dynamic Programming and Its Applications In Genomics and Machine Learning (10 credits)
  - Applications of Graph Algorithms in Genome Sequencing (10 credits)
  - Capstone Project in Genome Assembly (10 credits)

- **Proven motivator to complete the course.** MOOCs data tells that verified learners complete courses at a much higher rate than audit learners.

11 Forum

The forum can be found under the Discussion tab from the course home page.

Forum Rules

- Follow etiquette rules: respect other learners, make your post valuable for others (in particular, before asking check whether someone else has already asked it; keep your post as short as possible). See more common sense rules to follow.
- Please do not post solutions of programming challenges or their algorithmic parts (this violates the edX honor code), even if they do not pass the tests.

- We encourage you to post starter files for various programming languages. A starter file should only read the input data and write the output data.
- If your first attempt to solve a programming challenge failed, and then you debugged and fixed your program, you may want to share this bug. Examples: "Remember to use // for integer division for Python3", "Remember that a class name and a file name should match for Java", "Remember to use the long long type if you are dealing with large numbers for C++", but please do not post the code, just mention the "idea" of the bug.
- You may want also to help other learners by posting small tests that can help to catch a bug. In this case, please comment how did you come up with the test, so that others can learn from your creativity.
- Please note that instructors are not going to reply to the following typical questions.
  - *"My program runs fine on my local machine, but fails in the grader. Could you fix the grader?"*
    All the learners' computers are slightly different, so the only way to make grading fully objective is to grade all solutions on the same server. Sometimes, the result of compiling and running the same program is different on different computers because of different compilers, different compiler settings, different operating system or just some bugs in the program code that lead to undefined behavior. However, it is always possible to write a correct program that works correctly both on your local machine and in our grader. You will have to write such a program to pass. We do our best to specify all the important parameters of the grader that we know of here.
  - *"I use exactly the same compiler and flags as in the grader, but my program has different outputs on my machine and in the grader. Could you check the grader?"*
    This usually happens with buggy programs that run into undefined behavior.
  - *"Any hints about test 42?"*
    Recall that we hide the test cases intentionally. Please test and stress test you program to fix it. If you have already tested a lot (considered all corner cases that you can imagine, constructed a set of manual test cases, applied stress testing), but your program still fails and you are stuck, try to ask for help on the forum. We encourage you to do this by first explaining what kind of corner cases you have already considered (it may happen that when writing such a post you will realize that you missed some corner cases!) and only then asking other learners to give you more ideas for tests cases.
  - *"How to compile/run a starter file?"*
    Please note that though this class has many programming exercises, it is not a programming class. We expect you to know how to program in a programming language of your choice.