HOSTOS COMMUNITY COLLEGE
DEPARTMENT OF MATHEMATICS

COURSE: CSC 320 COMPUTER ALGORITHMS

CREDIT HOURS: 3.0

EQUATED HOURS: 3.0

CLASS HOURS: 3.0

PRE/COREQUISITE: ENG 111 and CSC 275 Students are expected to have experience in a high-level programming language.

RECOMMENDED TEXT: Introduction to Algorithms, 3rd edition, by Cormen, Leiserson, Rivest, and Stein,
SUGGESTED SOFTWARE:
MyLab Programming with Pearson eText -- Access Code Card -- for C++ How to Program (Early Objects Version)

REFERENCE MATERIAL

Khan Academy algorithm course:
https://www.khanacademy.org/computing/computer-science/algorithms
MIT Open Courseware with video lectures:
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/syllabus/

CUNY Blackboard: All course-related assignments and material will be available on the blackboard. Announcements will be posted on the blackboard. Students must have a working CUNY portal account in order to receive course-related information

DESCRIPTION: This course is an introductory undergraduate course on the design and analysis of algorithms. The goal of this course is to introduce basic- fundamental algorithm design techniques that are interesting both from a theoretical and practical point of view. We will cover basic algorithm design techniques such as divide-and-conquer, dynamic programming, and greedy techniques for optimization. We will cover techniques for proof of the correctness of algorithms, and also asymptotic analysis of algorithm time bounds by the solution of recurrence equations. Some specific algorithm topics include: deterministic and randomized sorting and searching algorithms, depth and breadth first search graph algorithms for finding paths and

matchings, and algebraic algorithms for fast multiplication and linear system solving. Student computer background should include recursive procedures, and data structures from object orientated programming such as arrays, queues, binary trees, graphs, and linked lists. Student mathematical background should include mathematical induction and elements of calculus. This course will employ pseudo code for which an understanding of a high-level programming course is required.

EXAMS /PROJECTS: 2 Exams, 2 projects, and a comprehensive final exam

GRADES: A, A-, B+, B, B-, C+, C, D, I, F.

GRADING

Homework	10%
Projects (2)	20%
Exams (2)	40%
Final Exam	30%

STUDENT LEARNING OUTCOMES:
Students who complete the course will have demonstrated the ability to do the following:

1. Analyze the asymptotic performance of algorithms. This includes:

- Argue the correctness of algorithms using inductive proofs and loop invariants.
- Analyze worst-case running times of algorithms using asymptotic analysis. Compare the asymptotic behaviors of functions obtained by the elementary composition of polynomials, exponentials, and logarithmic functions. Describe the relative merits of worst-, average-, and best-case analysis.
- Analyze average-case running times of algorithms whose running time is probabilistic. Employ indicator random variables and linearity of expectation to perform the analyses. Recite analyses of algorithms that employ this method of analysis.

2. Apply important algorithmic design paradigms and methods of analysis. This includes:

- Explain the basic properties of randomized algorithms and methods for analyzing them. Recite algorithms that employ randomization. Explain the difference between a randomized algorithm and an algorithm with probabilistic inputs.
- Analyze algorithms using amortized analysis, when appropriate. Recite analyses of simple algorithms that employ this method of analysis. Describe different strategies for amortized analysis, including the accounting method and the potential method.

3. Demonstrate familiarity with major algorithms and data structures.

- Describe the divide-and-conquer paradigm and explain when an algorithmic design situation calls for it. Recite algorithms that employ this paradigm. Synthesize divide-and-conquer algorithms. Derive and solve recurrences describing the performance of divide-and-conquer algorithms.

- Describe the dynamic-programming paradigm and explain when an algorithmic design situation calls for it. Recite algorithms that employ this paradigm. Synthesize dynamic-programming algorithms, and analyze them.
- Describe the greedy paradigm and explain when an algorithmic design situation calls for it. Recite algorithms that employ this paradigm. Synthesize greedy algorithms, and analyze them.
- Explain the major algorithms for sorting. Recite the analyses of these algorithms and the design strategies that the algorithms embody. Synthesize algorithms that employ sorting as a subprocedure. Derive lower bounds on the running time of comparison-sorting algorithms, and explain how these bounds can be overcome.

4. Synthesize efficient algorithms in common scientific design situations.

- Explain the major elementary data structures for implementing dynamic sets and the analyses of operations performed on them. Recite algorithms that employ data structures and how their performance depends on the choice of data structure. Synthesize new data structures by augmenting existing data structures. Synthesize algorithms that employ data structures as key components.
- Explain the major graph algorithms and their analyses. Employ graphs to model engineering problems, when appropriate. Synthesize new graph algorithms and algorithms that employ graph computations as key components, and analyze them.
- Demonstrate familiarity with applied algorithmic settings - such as computational geometry, operations research, security and cryptography, parallel and distributed computing, operating systems, and computer architecture - by reciting several algorithms of importance to different fields.

COURSE OUTLINE

1. Overview of the course: Introduction to Algorithms
2. Analyzing and Designing Algorithms: Ex. Merge and Insertion and Bubble Sort Algorithms
3. Basic Structures in Discrete Math and Data Structures::Review of selected topics (2 lectures)
4. Functions: Function Notation Standard Notation Ex. Growth Function (2 lectures)
5. Recursive Equations: Ex. Substitution Method, Recursive Tree Method, Master Method, Divide and Conquer Method, MergeSort (2 lectures)
6. Sorting Methods: Lower Bounds in Sorting, Counting Sort Ex. Radix and Bucket Sort Methods
7. Review
8. Exam 1 (11th lecture)
9. Binary Search Trees, Querying in a Binary Search Tree, Insertion and Deletion in a Binary Search Trees.
10. Red-Black Search Trees: Properties, Insertion, and Deletion in Red-Black Trees
11. Randomized Trees and Quicksort: Description, Performance, and Analysis of Quicksort, Randomized Version of Quicksort.
12. Medians and Order Statistics-Selection: Minimum and Maximum, Selection in Expected Linear Time, Selection in Worst Case Linear Time.
13. Hashing Algorithm: Direct Access and Hast Tables, Hash Functions Open Addressing

and Perfect Hashing.

14. Algorithms and Number Theory: Concepts in Elementary Number Theory, Greatest Common Divisor, Modular Arithmetic, Solving Modular Linear Equations, The Chinese Remainder Theorem, Powers of an Element,
15. Cryptography Algorithms: The RSA Public-Key Cryptosystem, Primality Testing, Integer Factorization.
16. Topics in Pattern String Matching: Selected Topics may include: The Naïve-String-Matching Algorithm, The Rabin-Karp Algorithm, String Matching with Finite Automata, The Knuth-Morris-Pratt Algorithm.
17. Elementary Graph Algorithms: Representation of graphs, Breadth-First Search, Depth-First Search, Topological Sorting (2 lectures)
18. Review
19. Exam 2 (23rd Lecture)
20. Greedy Algorithms: Introduction to Greedy Algorithm e.g., Activity Selection, Elements of the Greedy Algorithm, Huffman Codes, Theoretical Foundations for Greedy Algorithms, Ex. Task Scheduling. (2 lectures)
21. Dynamic Programming and Graph Algorithms: Assembly-Line Scheduling, Matrix-Chain Multiplication, Elements of Dynamic Programming, Longest Common Subsequence, Optimal Binary Search Trees. Graph Algorithms: Single Source Shortest Path-Dijkstra's Algorithm, Multiple Source Shortest Path & Maximum Flow – Ford Fulkerson Method (2 lectures)
22. Review
23. Final Exam (29th Lecture)