

THINK BIG WE DO™



CSC 212

Data Structures & Algorithms

Fall 2022 | Jonathan Schrader

Final Project

Breakdown

Groups of 3-4 students

- Groups of size 4 will be expected to produce more

Deliverables include:

- source code submitted (GitHub)
- class presentation (Gradescope/Zoom)
- project report (Gradescope)

Source Code (20%)

Code Repository must be created on GitHub

- this will be considered as your submission (no need to upload source code to gradescope)
- individual contributions must be clear on the repo

Code Organization (graded)

- use of classes and functions encouraged
- proper indentation
- use of comments

TAs may be asked to compile your code locally (graded)

- must create a markdown *README* in your GitHub repo with instructions on dependencies and how to compile your code
- if project is incomplete, partial credit may be given after TAs analyze your code

Presentation (40%)

Team will present for 15 minutes to the class (Zoom)

- explain the data structure/algorithm your team is focusing on (8 min)
- show your source code (running) and findings (plots) (4 min)
- answer questions (3)

Partial credit may be given if the project is incomplete but students show a very good understanding of the data structure/algorithm

Submit the PDF of your presentation to Gradescope

% of your presentation grade will come from observing other group's presentation

Project Report (40%)

Report must include:

- title of project
- names of all team members
- introduction (explain the context and purpose of your project)
- methods (explain the data structure/algorithm with more details)
- implementation (describe your work, include plots or figures if necessary)
- contributions (describe precisely what each member contributed to the project)

Partial credit may be given if the project is incomplete but the report includes relevant information regarding your progress

Submit the PDF of your report to Gradescope

Topics



Sorting Algorithms

(limit = 3)

Implement the following algorithms

- insertion sort
- mergesort
- quicksort
- 1 additional algorithm not present in class

Benchmark all algorithms recording their runtime

- use different sequence sizes
- use different types of sequences: sorted, reversed, random, partially sorted

Left-leaning red-black BSTs

(limit = 2)

Implement the Data Structure (using classes)

Must store the words of an input text file

- for the word (*node*) in the tree, a count with the number of repeats must also be stored

Ensure at least insert/search methods are correctly implemented

- search must also return/show the count

Generate a DOT file for visualization

B-Tree

(limit = 2)

Implement the Data Structure (using classes)

Must store the words of an input text file

- for the word (*node*) in the tree, a count with the number of repeats must also be stored

Ensure at least insert/search methods are correctly implemented

- search must also return/show the count

Generate a DOT file for visualization

Trie

(limit = 2)

Implement the Data Structure (using classes)

Must store the words of an input text file

- for the word (*node*) in the tree, a count with the number of repeats must also be stored

Ensure at least insert/search methods are correctly implemented

- search must also return/show the count

Generate a DOT file for visualization

Splay Tree

(limit = 2)

Implement the Data Structure (using classes)

Must store the words of an input text file

- for the word (*node*) in the tree, a count with the number of repeats must also be stored

Ensure at least insert/search methods are correctly implemented

- search must also return/show the count

Generate a DOT file for visualization

Segment Tree

(limit = 2)

Implement the Data Structure (using classes)

Must read intervals from an input text file

Ensure at least insert/search methods are correctly

Generate a DOT file for visualization

k-d Tree

(limit = 2)

Implement the Data Structure (using classes)

Must read data from an input text file

Ensure at least insert/search methods are correctly

Generate a DOT file for visualization

Sparse Matrices with Linked Lists

(limit = 3)

Load sparse matrices from files

Design a class for representing a matrix using linked lists

- memory efficient

Implement at least matrix addition and matrix multiplication

String Search I

(limit = 2)

Implement the following algorithms:

- rabin-karp
- boyer-moore

Benchmark all algorithms recording their runtime

- generate plots of running time of multiple sizes

String Search II

(limit = 2)

Implement the following algorithms:

- knuth-morris-pratt
- boyer-moore

Benchmark all algorithms recording their runtime

- generate plots of running time of multiple sizes

Recursive Graphics

(limit = 2)

Implement the following recursive fractals:

- Sierpinski Triangle
- Hilbert Curve
- Koch snowflake

Save the output to a PDF or an image file

Convex Hull

(limit = 2)

Implement the graham's scan algorithm

Read a set of points from a file

Find a way to save a visualization

- can generate a DOT file