Comp 460 - Algorithms & Complexity

Spring Semester 2020 - Week 1

Dr Nick Hayward

Course Details

Lecturer

- Name: Dr Nick Hayward
- Office hours
  - *Tuesday by appointment*
- Faculty Page

Important dates for this semester

- Project outline and mockup - presentation & demo
    - *Tuesday 11th February 2020 @ 4.15pm*
- Spring break
    - ***n.b.*** *no formal class: Tuesday 3rd March 2020*
- DEV week: 10th to 17th March 2020
- DEV week - presentation & demo
    - *17th March 2020 @ 4.15pm*
- Final class: 21st April 2020
- Final presentation & demo
    - *21st April 2020 @ 4.15pm*
- Exam week: 27th April to 2nd May 2020
    - *Final assessment due on 28th April 2020*

Coursework schedule

Presentations, reports &c.

- project outline and mockup
  - *due Tuesday 11th February 2020 @ 4.15pm*
- DEV week demo
  - *due Tuesday 17th March 2020 @ 4.15pm*
- final team demo
  - *due Tuesday 21st April 2020 @ 4.15pm*
- final team report
  - *due Tuesday 28th April 2020*

Initial Course Plan - Part 1

(up to ~ DEV Week)

- intro & review of fundamental concepts
  - *algorithms and data structures*
  - *app development and design patterns*
  - *initial testing of performance*
  - *...*

Initial Course Plan - Part 2

(Up to the end of the semester)

- detailed review of applied usage
  - *algorithms & data structures*
  - *practical use & applications*
  - *app testing and performance*
- integration of algorithms and data structures
  - *problem solving*
  - *app usage*
  - *app context*
  - *...*

Assignments and Coursework

Course will include

- weekly bibliography and reading (where applicable)
- weekly notes, examples, extras...

Coursework will include

- exercises, fun quizzes, and discussions (Total = 40%)
  - *various individual or group exercises and discussions*
- project outline & mockup (Total = 15%)
  - *brief group presentation of initial concept and mockup*
  - *due Tuesday 11th February 2020 @ 4.15pm*
- DEV week assessment (Total = 15%)
  - *DEV week: 10th to 19th March 2020*
  - *presentation & demo: Tuesday 17th March 2020 @ 4.15pm*
- end of semester final assessment (Total = 30%)
  - *demo due Tuesday 21st April 2020 @ 4.15pm*
  - *final report due Tuesday 28th April 2020 @ 4.15pm*

Exercises, fun quizzes, & discussions

Course total = 40%

- exercises
  - *help develop course project*
  - *test course knowledge at each stage*
  - *get feedback on project work*
- discussions
  - *sample problems, articles, applications...*
  - *various contextual concepts and material*
- fun quizzes
  - *various quizzes to reinforce course material*
- extras
  - *code and application reviews*
  - *various other assessments*
  - *peer review of demos*

Course total = 60% (Parts 1, 2 and 3 combined)

Initial overview

- combination project work
  - *part 1 = project outline & mockup (15%)*
  - *part 2 = DEV Week development & demo (15%)*
  - *part 3 = final demo and report (30%)*
- group project (max. 4 persons per group)
- design and develop an app
  - *purpose, scope &c. is group's choice*
    - NO blogs, to-do lists, note-taking...
    - chosen topic requires approval
    - examples apps include
      - mobile
      - gaming
      - desktop
      - web
      - terminal
  - *must implement algorithms & data structures*

Project outline & mockup assessment

Course total = 15%

- begin outline and design of an application
  - *built from scratch - languages include*
    - JavaScript
    - Python
    - C
    - ...
  - *builds upon examples, technology outlined during first part of semester*
  - *must implement algorithms & data structures*
  - *purpose, scope &c. is group's choice*
  - *NO blogs, to-do lists, note-taking...*
    - chosen topic requires approval
  - *presentation should include mockup designs and concepts*

Project mockup demo

Assessment will include the following:

- brief presentation or demonstration of current project work
  - *~ 5 to 10 minutes per group*
  - *analysis of work conducted so far*
  - *presentation and demonstration*
    - outline current state of app concept and design
    - show prototypes and designs
  - *due Tuesday 11th February 2020 @ 4.15pm*

DEV Week Assessment

Course total = 15%

- continue development of application
  - *built from scratch*
  - *continue design and development of initial project outline and design*
  - *working app (as close as possible...)*
  - *NO blogs, to-do lists, note-taking...*
  - *...*
- outline research conducted
- describe data chosen for application
- define algorithms and data structures used in app
  - *why choose these options?*
  - *how have they been used?*
  - *define current performance &c.?*
  - *define testing of implementation & usage*
- show any prototypes, patterns, and designs

DEV Week Demo

DEV week assessment will include the following:

- brief presentation or demonstration of current project work
  - *~ 5 to 10 minutes per group*
  - *analysis of work conducted so far*
    - e.g. during semester & DEV week
  - *presentation and demonstration*
    - outline current state of app
    - explain what works & does not work
    - show implemented designs since project outline & mockup
    - show latest designs and updates
  - *due Tuesday 17th March 2020 @ 4.15pm*

Final Assessment

Course total = 30%

- continue to develop your app concept and prototypes
  - *working app*
    - must implement algorithms and data structures
  - *explain design decisions*
    - describe patterns used in design and development of app
    - structures, organisation of code and logic
  - *explain testing and analysis*
  - *show and explain implemented differences from DEV week*
    - where and why did you update the app?
    - perceived benefits of the updates?
  - *how did you respond to peer review?*
  - *...*
- final demo
  - *due Tuesday 21st April 2020 @ 4.15pm*
- final report
  - *due Tuesday 28th April 2020*

Goals of the course

A guide to developing applications with algorithms and data structures.

Course will provide

- guide to algorithms and data structures
- guide to developing application structures and patterns from scratch
- integrating algorithms and data structures to solve problems
- best practices and guidelines for development
- fundamentals of application development
  - *practical algorithms and data structures*
- intro to advanced options for app development
- ...

Course Resources - part 1

Website

Course website is available at https://csteach460.github.io

- timetable
- course overview
- course blog
- weekly assignments & coursework
- bibliography
- links & resources
- notes & material

No Sakai

Course Resources - part 2

GitHub

- course repositories available at https://github.com/csteach460
  - *weekly notes*
  - *examples*
  - *source code (where applicable)*

Trello group

- group for weekly assignments, DEV week posts, &c.
- Trello group - ‘COMP 460 - Spring 2020 @ LUC’
  - *https://trello.com/csteach460*

Slack group

- group for class communication, weekly discussions, questions, &c.
- Slack group - ‘COMP 460 - Spring 2020 @ LUC’
  - *https://csteach460-2020.slack.com*

Group projects

- add project details to course's Trello group, *COMP 460 - Spring 2020 @ LUC*
  - *Week 1 - Project Details*
  - *https://trello.com/b/S5OAaKac/week-1-project-details*
- create channels on Slack for group communication
  - *please add me to the private channel*
- start working on an idea for your project
- plan weekly development up to and including DEV Week

Intro to Algorithms and Data Structures

- consider their usage in the context of application development
- includes algorithms and data structures
  - *may work together to solve defined problems*
- initially consider an algorithm as a way to solve problems
  - *data structures as a storage option*
- data structure will store the information associated with the problem
  - *work in tandem with the algorithm*
- common use for data structures and algorithms includes data sorting and searching
- basic to many structures
  - *e.g. stacks, queues, priority queues, bags &c.*
- then consider common algorithms for sorting, effective methods for organising data
  - *e.g. quicksort, mergesort, heapsort &c.*
- these algorithms and structures naturally help with search, including classic options
  - *e.g. binary search trees, hash tables &c.*
- may also form part of algorithms for advanced tasks, including
  - *graph traversal and searching*
  - *shortest path algorithms*
  - *minimum spanning trees*
  - *text manipulation and processing*
  - *data compression*
  - *...*

Why algorithms?

- noticeable benefit of algorithms
  - *their scope*
  - *application to many diverse disciplines*
  - *their inherent abstraction*
  - *…*
- broad range of uses, e.g.
  - *internet, science, social networks, video games, music…*
- used in almost every aspect of modern life and culture
- form an invaluable part of scientific research, art, and the humanities

A brief history of algorithms - part 1

- history of algorithms is fascinating to consider
- word itself, *algorithm*, has its roots in the 9th century
  - *with the mathematician Abdullah Muhammad bin Musa al-Khwarizmi*
  - *mathematician, scientist, and astronomer*
- *al-Khwarizmi* is often noted as the father of algebra
  - *the origin of today's word algorithm*
- 12th century Latin translation of a book by *al-Khwarizmi*
  - *provided a translation of his name as Algorithmi*
  - *various alternatives of this story, but the underlying origin is consistent*
- whilst the specific word *algorithm* began with this mathematician and translation
  - *general concept we now associate with an algorithm has ancient roots.*

A brief history of algorithms - part 1

ancient roots

- origin of the use of algorithms may be traced as far back as Babylonian and Egyptian mathematics
- Babylonian and Egyptian mathematics
  - *often considered within the same context of early mathematical usage*
  - *Babylonian system was, in many respects, more advanced*
- e.g. Babylonians were able to work with the following
  - *square and cube roots*
  - *Pythagorean triples - 1200 years before Pythagoras*
  - *knew of the existence of* `pi`
  - *the exponential function, e - possible basic understanding*
  - *solve some quadratics - even polynomials of degree 8*
  - *solve linear equations*
  - *handle measurement of circles*
  - *...*
- their mathematics was not rudimentary and basic
  - *concerned with algebra and not geometry*
  - *an interesting contrast with the later Ancient Greeks*
- Babylonians used a sexagesimal, or `base-60`, number system
  - *inherited from the Sumerians and Akkadians*

A brief history of algorithms - part 1

Babylonian numbers

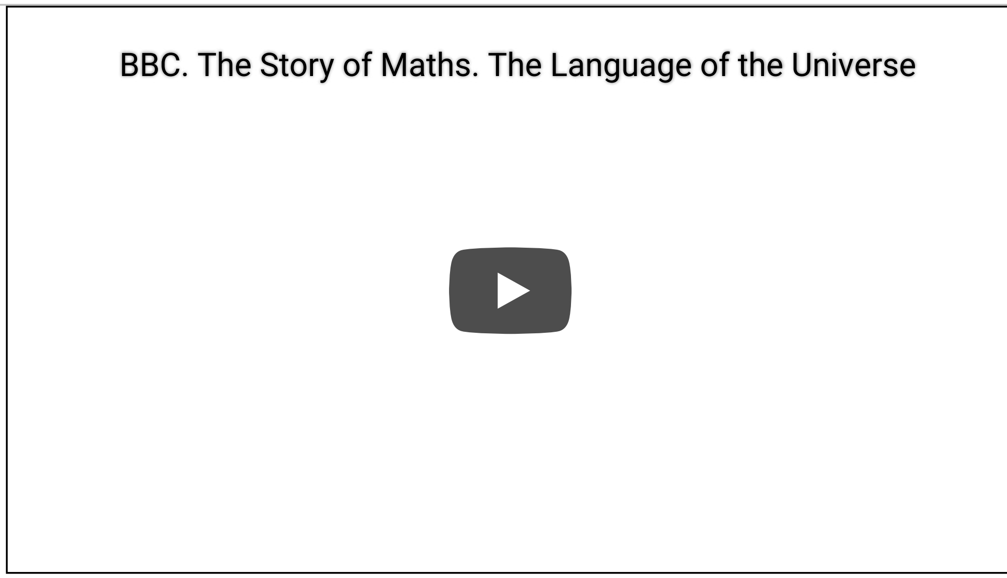
- sexagesimal number system adopted by the Babylonians
  - *used in conjunction with a place value*
  - *used to write numbers larger than 60*
- they had symbols for '1 to 59'
  - *then repeated these symbols in additional columns to represent larger numbers*
- simple example is as follows,

column 3	column 2	column 1
2	1	9

- gives us
  - *'$2(60^2) + 1(60) + 9 = 7269$'*
- we may see a basic algorithm at work for their underlying number system

Video - Mathematics

Babylonian numbers



The Story of Maths - Part 1

Source - Story of Maths - YouTube

A consideration of algorithms - part 1

- consider an algorithm as a series of instructions for completing a defined task
- all code that accepts an input, and provides a defined output
  - *may be considered analogous to an algorithm*
- e.g. these patterns may be seen in basic functions
  - *accept a parameter*
  - *commonly return a computed value*
- algorithms come in many different shapes and sizes
- e.g. we commonly use algorithms with
  - *search, graphs, AI and machine learning, gaming, and many more.*
- e.g. for gaming
  - *we might create an algorithm that allows mob objects to track and follow the player using graphs*
  - *might use k-nearest neighbor to define relationships with basic machine learning*
- as we review and develop example algorithms
  - *ommonly perform tests to determine performance, efficiency, speed, and comparative benefits*
  - *such runtime testing is commonly performed using Big-O notation*

A consideration of algorithms - part 2

- we shall cover the key ideas involved in designing algorithms
- how they depend on the design of suitable data structures
- how some structures and algorithms are more efficient than others for the same task
- we'll review a few basic tasks
  - *e.g. storing, sorting, and searching data*
- such tasks underpin much of computer science
  - *equally key to understanding the nature of algorithms*
- we may begin with some key data structures
  - *e.g. arrays, lists, queues, stacks, and trees*
- consider their use in a range of different searching and sorting algorithms
  - *leads to a consideration of efficient storage of data*
  - *e.g. in hash tables*
- also review various graph based representations
  - *covering necessary algorithms for efficient use, navigation, and manipulation*
- we'll investigate computational efficiency of such algorithms
  - *gaining insights on the pros and cons of the various approaches for each task*
- implementing various data structures and algorithms not restricted to particular programming languages
  - *examples will initially be defined in simple pseudocode*
  - *then implemented, where appropriate, in a chosen language*

A consideration of algorithms - part 3

- consider a general search problem, we might initially frame it as
  - *a search problem may be defined as a problem*
  - *requires finding a specific value, a target*
  - *within a space of potential values, a search space*
- we may define a suitable algorithm in the context of target, search space, & search algorithm
- *Target*
  - *piece of data you're searching for...*
  - *target can be either a specific value or a criterion that signifies successful completion of a search*
- *Search space*
  - *set of all possibilities to test for the target*
  - *e.g. search space could be a list of values or all the nodes in a graph*
  - *a single possibility within the search space is called a state*
- *Search algorithm*
  - *set of specific steps or instructions for conducting the search*
- some algorithms will, of course, require additional components, complexities, and considerations
  - *we now have a framing we may use*
  - *e.g. to begin reviewing solutions to the problem*

Algorithms and programs - part 1

> *a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time*

- an algorithm must be precise enough to be understood by us - developers and programmers
- in order to be executed by a computer
  - *generally need a program that is written in a rigorous formal language*
- often a key consideration as we design and test algorithms
- need to define the algorithm abstracted
  - *separate from formalities and depth necessary for a formal implementation in a programming language*
- try to initially reduce the baggage associated with a coded example
- also need to consider relevance of different programming paradigms
  - *e.g. imperative vs declarative*
- Imperative programming
  - *describes computation in terms of instructions that change the program/data state*
  - *common example with JavaScript and direct DOM manipulation*
- Declarative programming
  - *specifies what the program should accomplish without describing how to do it*
  - *e.g. React JS*

Algorithms and programs - part 2

- subtle difference in the examples
  - *imperative commonly defines step by step instructions*
  - *e.g. directly updating an element in the DOM*
  - *declarative with React may define how the element should look, act &c.*
  - *but it doesn't directly update the DOM for the element*
  - *simply defines how it should be rendered &c. for a given state in the app*
- with *declarative*
  - *should not think about how to do achieve a specific result*
  - *instead, consider the result from a given update to state*
- commonly easier to initially understand algorithm design from an *imperative programming* perspective
  - *pseudocode helps us consider this approach without the formal baggage of a given programming language.*
- once a design has been implemented
  - *we may code an example in a chosen programming language*

Algorithms and programs - part 3

code examples

- imperative - plain JavaScript

```
const group = document.getElementById('group');
const btn = document.createElement('button');

btn.className = 'btn red';

btn.onclick = function(event) {
 if (this.classList.contains(‘red’)) {
   this.classList.remove(‘red’);
   this.classList.add(‘blue’);
 } else {
   this.classList.remove(‘blue’);
   this.classList.add(‘red’);
 }
};
container.appendChild(btn);
```

- declarative - React JS

```
class Button extends React.Component{
  this.state = { color: 'red' }
  handleChange = () => {
    const color = this.state.color === 'red' ? 'blue' : 'red';
    this.setState({ color });
  }
  render() {
    return (<div>
      <button
         className=`btn ${this.state.color}`
         onClick={this.handleChange}>
      </button>
    </div>);
  }
}
```

Resources

- Babylonian Mathematics
- Declarative programming
- Imperative programming
- React JS
- Story of Maths - YouTube