COMP251: Algorithms and Data Structures

Jérôme Waldispühl
School of Computer Science
McGill University
About Me

• Jérôme Waldispühl
• Associate Professor of Computer Science
• I am conducting research in Bioinformatics

• How to reach me?
  o Office hours (TBA)
  o By appointment (email me to schedule a meeting)
  o Email: cs251@cs.mcgill.ca

(Note: This will be the only email address you should use and from which you can expect an answer)
Where to get announcements & updates?

Official channel:

• Course web page:  

Other channels

• MyCourses
• In class!
Evaluation

- 40% for 5 assignments
- 10% for quizzes*
- 49% for the final exam
- 1% for agreeing and following course policy

* The quizzes are optional. If the average grade of the quizzes is lower than your score at the final exam, the weight of the quizzes will be reported to the final (i.e. your final will count for 60% of your grade).

Warning: There will be no modification of this scheme.
Schedule

• Classes start... Today
• Classes end on November 28 (included).
• No midterm.
• Final exam: TBD
Organization

- Lecture are recorded and available for streaming but cannot be downloaded.
- Slides available on the course webpage.
- Overflow room is expected to close at end of Week 1.
- Check schedule online to find TA office hours.
- Programming questions have to be asked to TAs, but do not expect TAs to debug your code for you.
- We have tutorials (e.g. Sep 25).
- How to reach us? cs251@cs.mcgill.ca
Outline

- Sep 4-6: Background & Review
- Sep 11: Hash tables
- Sep 13 - 27: Tree ADT
- Sep 25: Tutorial
- Oct 2\textsuperscript{nd}: Intro to Algorithm design (Greedy algorithm)
- Oct 4 - 25: Graph algorithm
- Oct 30 - Nov 13: Algorithm design
- Nov 15 - 22: Algorithm analysis & randomized algorithms
- Nov 27 - 29: Review
What is different from COMP250?

- Algorithm design
- Proofs (correctness)
- Analysis of performance

Requirement: Basic understanding of probabilities and discrete math (See co-requisites).
Textbooks

(Available as E-book at the McGill library)


Textbooks are recommended but not mandatory.
Assignments

• Only programming questions
• Programming Language: Java
• Read carefully the formatting guidelines.
• Strictly follow the formatting guidelines.
• Submit your answers electronically on MyCourse.
• Each file must be submitted separately. Do not zip your files!
• Re-submission accepted before the deadline.
• Discuss the assignment, but do not share/copy solutions.
• Write (in comments) the name of persons with whom you discussed/collaborated (including instructor and TA).
Answering programming questions

• Submit the Java source file (i.e. not class file)
• Separate MyCourses folder
• Ensure that your files compile on SOCS workstations (Note: Create an account if you do not already have one)
• Indent and comment your code!
• Use the template provided.
• Do not use custom libraries (unless specified).
• Follow the syntax of the command line provided in the question.
• Use the test input & output (Note: It does not guarantee that your program is 100% correct).
Tips

• We will provide test cases (i.e. input & output). Use them!

• BUT note that **passing all tests does not guarantee that your program is 100% correct**. We will use different ones to check that your program is correct.

• It is your responsibility to guarantee that your program:
  • Compile on SOCS workstations
  • Run with the proper syntax of the command line
  • Produces an output that follows the guidelines

• Submit early. We will run the grader ~24h before the deadline. It will give you an opportunity to fix your assignment and resubmit before the deadline.
Policy

• Late assignments will receive a 20% penalty if they are returned within less than 24h after the end of the deadline. They will not be graded afterward. (Tip: Submit preliminary versions early)

• The only exceptions will be medical exceptions. You must provide a medical note (instructor & McGill).

• No plagiarism!
Next class

- Review of COMP 250 material
  Take online quiz (Practice only. not graded! Link available tomorrow on course website)

- Basic probability (expectation, indicator)
Binary numbers

- Definition
- Conversion decimal $\leftrightarrow$ binary
- Addition
- Size of the representation
Recursions

- Definition (recursive case & base case)
- Binary search
- Fibonacci
- Merge Sort
- How to write a function describing the running time of a recursive algorithms.
- Estimate the number of recursive calls.
- Dividing original problem into roughly equal size subproblems usually gives better running times.
Running time and big-Oh

• Running time:
  o Counting primitive operations
  o Dealing with loops: $\sum_{i=1}^{n} i = n(n+1)/2$ is $O(n^2)$
  o Worst-case vs average-case vs best-case
• Big-Oh notation:
  o Mathematical definition
  o Big-Oh is relevant only for large inputs. For small inputs, big-Oh may be irrelevant (remember integer multiplications)
• Big-Theta, Big-Omega
• Unless mentioned otherwise, big-Oh running time is for worst-case.
• You need to know and understand the big-Oh running time of all algorithms seen in class and in homeworks.
Data Structures

• **Array**
  - running time for insert, delete, find...

• **Single-linked list**
  - Better than arrays:
    - Easier to insert and delete
    - No need to know size in advance
  - Worse than arrays:
    - finding the n-th element is slow (so binarySearch is hard)
    - Require more memory (for the "next" member)

• **Doubly-linked list**
  - Allow to move backward
  - Makes deleting elements easier

• **Stacks and queues**
  - You should understand all applications we saw
ADT (Abstract Data Structure)

What it is?

Description of the *interface* of a data structure. It specifies:

- What type of data can be stored
- What kind of operations can be performed
- Hides the details of implementation

Why it is important?

Simplifies the way we think of large programs
Trees

• treeNode representation
• Vocabulary: node, leaf, root, parent, sibling, descendants, ancestors, subtree rooted at x, internal and external nodes, ordered, binary, proper binary
• Depth and height
  – Definition
  – How to compute it.
• Tree traversal
  – Pre-order, In-order, Post-order
Dictionary ADTs

• Stores pairs (key, info)
• Operations: find(key), insert(key, info), remove(key)
• Cases where array implementation is bad (with complexity)
• Cases where linked-list implementation is bad (with complexity)
Dictionary ADTs with Binary Search trees (BST)

• Property: for any node $x$,
  – keys in the left subtree of $x$ have keys smaller or equal to $\text{key}(x)$ and
  – keys in the right subtree of $x$ have keys larger or equal to $\text{key}(x)$

• Algorithm to find a key and its running time $O(h) = O(\log n)$ if the tree is balanced.

• Inserting a new key. Running time $O(h)$. Sequence of insertion that can lead to bad running times.

• Removing a key.

• You need to be able to execute these algorithms by hand on examples.
Dictionary ADTs with Hash Tables

• Implements a dictionary
• Idea:
  – map keys to buckets
  – Each bucket is itself a dictionary
• Hash functions:
  – Goal: minimize collisions
  – Easy to compute
• Best case:
  – keys are distributed uniformly among the buckets. Each bucket contains few keys
• Worst case:
  – All keys end-up in the same bucket
Priority queues

• Heap property:
  – key(x) is smaller or equal to the keys of children of x.
  – All h-1 first levels are full, and in the last level, nodes are packed to the left

• Operations:
  – findMin(). Algorithm. O(1)
  – insert(key). Bubbling-up. O(log n)
  – removeMin(). Bubbling-down. O(log n)

• Array representation of heaps

• HeapSort
  – insert keys one by one
  – removeMin() one by one
Graphs

• All the terminology
• Data structures for representing graphs:
  – Adjacency-list
  – Adjacency-matrix
  – Running time of basic operations with each data structure
• Graph traversal
  – Depth-first search
    • Recursive
    • Iterative using a stack
  – Breadth-first search
    • Iterative using a queue
• IMPORTANT:
  Applications of DFS and BFS