COMP251: Algorithms and Data Structures

Jérôme Waldispühl
School of Computer Science
McGill University
About Me

• Jérôme Waldispühl
• Associate Professor of Computer Science
• Research in Bioinformatics & Human-Computing

• 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: 
  
  \[\text{http://www.cs.mcgill.ca/~jeromew/comp251.html}\]

Other channels

• MyCourses
• Reddit: \[\text{https://www.reddit.com/r/mcgillCOMP251/}\]
• In class!
General inquiries

Best option: Email cs251@cs.mcgill.ca

Who read your email? Instructor and head TA.

Who will answer to your email? Usually the Head TA but sometimes the instructor too.

FAQ: “I am not satisfied with the answer from the head TA. Can I email the instructor directly?”

Answer: You can, but the answer you will get is very unlikely to be different. Why? The answer you received has already been validated by the instructor. You should treat answers from the Head TA as answers from the instructor.
Evaluation Scheme

• 24% for 4 assignment (6% each)
• 25% for the mid-term exam*
• 50% for the final exam
• 1% for agreeing and following course policy

* If you perform better at the final than at the mid-term, the score of your final will replace the one of your mid-term.

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

• Classes start... Today
• Last class: November 28
• Tutorials: September
• Assignments (subject to changes):

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Topic</th>
<th>Released on</th>
<th>Due on</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hashing/trees</td>
<td>Sep 24</td>
<td>Oct 8</td>
</tr>
<tr>
<td>2</td>
<td>Graph algo</td>
<td>Oct 8</td>
<td>Oct 22</td>
</tr>
<tr>
<td>3</td>
<td>Flow nets</td>
<td>Oct 31</td>
<td>Nov 14</td>
</tr>
<tr>
<td>4</td>
<td>Algo design</td>
<td>Nov 14</td>
<td>Nov 28</td>
</tr>
</tbody>
</table>

• Midterm: 2nd part of October (TBD, possibly in the evening)
• Final exam: December (TBD)
Miscellaneous

• Lectures are recorded and available for streaming but cannot be downloaded (Note: This is not a substitute for not attending classes).
• Slides available on the course webpage.
• Check schedule on the course webpage for TA office hours.
• Programming questions can be asked to TAs, but do not expect them to debug your code.
• We will organize Java tutorials.
• How to reach us? cs251@cs.mcgill.ca
Outline

- Sep 3-5: Background & Review
- Sep 10: Hash tables
- Sep 12 - 26: Tree ADT
- Sep 24-26: Tutorial
- Oct 1: Intro to Algorithm design (Greedy algorithm)
- Oct 3 - 24: Graph algorithm
- Oct 29 - Nov 12: Algorithm design
- Nov 14 - 21: Algorithm analysis & randomized algorithms
- Nov 26 - 28: Review
What is different from COMP250?

Topics:

• Algorithm design
• Proofs (correctness)
• Analysis of performance
• Programming (if you took COMP250 last term)

Requirements:

• Basic understanding of probabilities and discrete math
• Programming in Java
Textbooks

(Available as E-book at the McGill library)


Textbooks are recommended but not mandatory.
Assignments

• Programming questions + Proof + Quizzes
• Programming Language: Java
• Read carefully the formatting guidelines.
• Strictly follow the formatting guidelines.
• Written answers: PDFs generated by LaTeX, Word or equivalent. Scans from Uprint accepted but not pictures from phone
• Submit your answers electronically on MyCourse.
• Do not zip your files! Submit each file separately.
• You can replace your files before the deadline.
• Discuss but do not share/copy solutions.
• Indicate the name of persons with whom you discussed/collaborated (including teaching staff).
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).
Automated grading

“His assignments were not incredibly difficult, but you would get a 0 for tiny, unimportant details. I misnamed a file by one letter, automatic zero. After spending 3 weeks on the assignment, they made no exceptions, had no mercy or understanding.”

(Anonymous comment on ratemyprofessor.com)

1. Warning: Assignments were not including proofs
2. We (staff) only can decide what is important or not
3. In 2 weeks you have plenty of time to check the guidelines (a.k.a. do not wait the last minute)
4. No exception is the only way to be fair

Note: we also get nice comments sometimes
Plagiarism

1. We run programs to automatically detect possible cases of plagiarism in programming assignments.

2. We manually review each case.

3. If we consider there is a case of plagiarism, we report directly to a disciplinary officer. We send email notifications after.

4. The rest of the procedure is out of our hands. Still, you can consult a description of the disciplinary procedure at: https://www.mcgill.ca/students/srr/honest/staff/student
Questions about assignments?

About the assignment:
1. (Preferred) Go the office hours of TAs
2. Email cs251@cs.mcgill.ca

Note 1: We do not guarantee answers to emails sent less than 24h from the deadline.
Note 2: By default, we do not debug your code.

About the grading:
1. Go to the office hours of the TA who graded the assignment
2. Email cs251@cs.mcgill.ca after you discussed with the TA
Advices

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

• BUT 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.** (Advice: 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
Next Class

• Review of COMP 250
• Intro to Hashing
• Answer online survey before the class (link available on the course web site).