COMP 250: Review

Jérôme Waldispühl
School of Computer Science
McGill University
Final Exam

• April 17\textsuperscript{th} at 18h30.
• No Book, no calculator, no headphone, etc.
• 1 crib-sheet (A4 or letter format)
• 50\% of final grade (or 60\% if your final’s grade is higher than your average quizzes’ grades)
• Multiple choice exam. 32 questions. No penalty for wrong answers ⇒ If you do not know, try to guess.
Office hours

• Check calendar on the course webpage.
• Me:
  o Every day 3-4pm until Friday. More early next week.
  o Question about course material.
• Many more options with other TAs
How to prepare your exam?

• Check practice exam/questions posted on the course Web page.

• Check practice exam/questions posted by previous instructors of COMP 250.

• Email us ([cs250@cs.mcgill.ca](mailto:cs250@cs.mcgill.ca)) or come to OH with precise questions.

• Reading the slides is not enough. Multiple choices questions will require deeper understanding of the material.
Topics covered

• Lecture 1 to 31.
• Topics (lectures 32-35) will not be covered in the final. But the lectures are used to illustrate important concepts potentially in the final.
• No Java, but possibly pseudo-code questions
• No need to know the exact pseudo-code of the algorithms covered in class, but you need to understand their logic.
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.
Binary search

```
1 1 3 5 6 7 9 9
```

Search [0:7]

5 < 6 \(\Rightarrow\) look into right half of the array

```
1 1 3 5 6 7 9 9
```

Search [4:7]

7 > 6 \(\Rightarrow\) look into left half of the array

```
1 1 3 5 6 7 9 9
```

Search [4:4]

6 is found. Return 4 (index)
Recursion time

$T(n)$: Running time for an input of size $n$

For most recursive algorithms seen in class, we can write:
$T(n) = a \times T(n') + f(n)$

Where
- $a$: number of recursive calls
- $n'$: the size of the sub-problem to solve (e.g. $n/x$, $n-x$)
- $f(n)$: the running time (i.e. number of primitive operations) processed by the function.

Examples:
- Binary search: $T(n) = T(n/2) + 1$
- Merge sort: $T(n) = 2 \times T(n/2) + n$
Divide-and-conquer recurrence: proof by recursion tree

**Proposition.** If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n.$

$$T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2 \cdot T(n/2) + n & \text{otherwise}
\end{cases}$$

**Pf 1.**

- $n = n$
- $2 \cdot (n/2) = n$
- $4 \cdot (n/4) = n$
- $8 \cdot (n/8) = n$
- $\vdots$

$$T(n) = n \log_2 n$$

(Kleinberg & Tardos, 2005)
Number of recursive calls

Let \( T(n) = 3 \cdot T\left(\frac{2}{3} \cdot n\right) + n \) be an equation describing a recursive algorithm with a base case when \( n = 1 \) (\( n \) being the size of the input). We call \( n_k \) the size of the input of the \( k \geq 0 \) recursive call.

\[
\Rightarrow n_k = n \cdot \left(\frac{2}{3}\right)^k \\
\Rightarrow n \cdot \left(\frac{2}{3}\right)^k = 1 \\
\Rightarrow n = \left(\frac{3}{2}\right)^k \\
\Rightarrow \log(n) = k \cdot \log\left(\frac{3}{2}\right) \\
\Rightarrow k = \frac{\log(n)}{\log\left(\frac{3}{2}\right)} = \log_{\frac{3}{2}}(n)
\]
Proofs by Induction

• Review example made in class. Find more!
• Prove recursive case AND base case.
• Know basic formula:

\[
1 + 2 + 3 + 4 + 5 + \ldots + k = \sum_{k=1}^{n} k = \frac{n(n + 1)}{2}
\]

\[
1 + 2 + 4 + \ldots + 2^k = \sum_{k=1}^{n} 2^k = 2^{n+1} - 1
\]

\[
1 + x + x^2 + x^3 + x^4 + \ldots + x^k = \sum_{k=1}^{n} x^k = \frac{x^{k+1} - 1}{x - 1}
\]
Proof by induction

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise}
\end{cases}
\]

Pf 2. [by induction on $n$]
- Base case: when $n = 1$, $T(1) = 0$.
- Inductive hypothesis: assume $T(n) = n \log_2 n$.
- Goal: show that $T(2n) = 2n \log_2 (2n)$.

\[
T(2n) = 2T(n) + 2n \\
= 2n \log_2 n + 2n \\
= 2n (\log_2 (2n) - 1) + 2n \\
= 2n \log_2 (2n).
\]

(Kleinberg & Tardos, 2005)
Proof using loop invariants

Used to prove a for loop.

We must show:

1. **Initialization**: It is true prior to the first iteration of the loop.

2. **Maintenance**: If it is true before an iteration of the loop, it remains true before the next iteration.

3. **Termination**: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.
Quicksort

• What is the partition method doing, and how?
• Worst case: $O(n^2)$ (when?)
• Average case: $O(n \log n)$
Running time and big-Oh

- **Running time:**
  - Counting primitive operations
  - Dealing with loops: \( \sum_{i=1}^{n} i = n(n+1)/2 \) is \( O(n^2) \)
  - Worst-case vs average-case vs best-case

- **Big-Oh notation:**
  - Mathematical definition
  - 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
Next (last) lecture

- I will answer questions about the final and in general the material covered in this class.
- I will also give you an overview of opportunities after COMP250.