# CSC373

*Algorithm Design, Analysis & Complexity*

Sinan Li

2024

# CONTENTS

# Part I

# Notes

CHAPTER

# 1 INTRODUCTION

## 1.1 Course Information

- **Instructor**: Nathan Wiebe
  - **Email**: nawibe@cs.toronto.edu
  - **Office**: SF 3318C
- **Text**: [CLRS] *Introduction to Algorithms*: Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford
- **Disclaimer**: Many things are up in the air, so expect a somewhat bumpy ride at the start but hopefully, we will get through together! Use any of the feedback mediums (email, Piazza, ...) to let the instructor know if there are any suggestions for improvement.

## 1.2 Grading

### 1.2.1 Assignments

- **4 assignments**, best 3 out of 4
- Group work
  - In groups of *up to three* students
  - Best way to learn is for each member to try each problem
- Questions will be **more difficult**

- May need to mull them over for several days; do not expect to start and finish the assignment on the same day!
- May include bonus questions

- Submission on **crowdmark**, more details later. May need to compress the PDF.

### 1.2.2 Tests

- 2 term tests, one end-of-term test (final exam / assessment)
- Time and Place
  - Fridays during Tutorials
  - In-person

### 1.2.3 Grading Scheme

| | | | | |
|---|---|---|---|---|
| Best 3/4 Assignments | × | 10% | = | 30% |
| 2 Term Tests | × | 20% | = | 40% |
| Final Exam | × | 30% | = | 30% |

**Note**: There is **no** auto-fail policy for the final exam.

## 1.3 Course Information

### 1.3.1 What is this course about?

- What if we can't find an efficient algorithm for a problem?
  - Try to prove that the problem is hard
  - Formally establish complexity results
  - NP-completeness, NP-hardness, ...
- We'll often find that one problem may be easy, but its simple variants may suddenly become hard.
  - Minimum spanning tree (MST) vs. bounded degree MST
  - 2-colorability vs 3-colorability

### 1.3.2 Proofs

In this course you are expected to provide a clear and compelling argument about why you're right about ant claim about an algorithm. We call these argument proofs.

Proof structures used in this course:

- Induction
- Contradiction
- Desperation...

### Inductive Proof

Key idea with induction:

- Break the problem into a number of steps, $s(i)$.
- Show that induction hypothesis holds for base case $s(0)$.
- Show that if hypothesis holds for $s(i)$ then it holds for step $s(i+1)$.

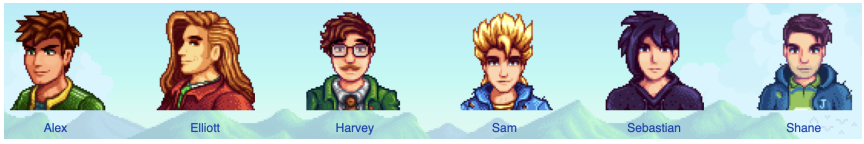
**Theorem 1.3.1 Principle of Mathematical Induction**

Let $P(n)$ be a predicate defined for integers $n \geq 0$. If

1. $P(0)$ is true, and
2. $P(k)$ implies $P(k+1)$ for all integers $k \geq 0$,

then $P(n)$ is true for all integers $n \geq 0$.

**Example.** Say you want to find the best person to marry in Stardew Valley.




You can apply a single iteration of Bubblesort to find that Sebastian is objectively the best person to marry.

```
1: procedure BUBBLESORT(A)
2:     for i = 1 to n − 1 do
3:         for j = 1 to n − i do
4:             if A[j] > A[j + 1] then
5:                 SWAP(A[j], A[j + 1])
6:             end if
7:         end for
8:     end for
9: end procedure
```

*Proof.* Proof by induction on the number of iterations of Bubblesort.

- **Base case**: $s(1)$

  Then swap doesn't happen and you have a trivially sorted array.

- **Induction Steps**: $s(i) \to s(i+1)$

  Assume that $s(i)$ is sorted by the algorithm for any array $s$ of length $i$.

  $$s_0, s_1, \ldots, y = s_{i-1}, x = s_i$$

  1. **Case 1**: $x > y$

     Then, comparison between $x$, $y$ says you should swap them.

  2. **Case 2**: $x \leq y$

     Then, comparison between $x$, $y$ says you should not swap them. The array is still sorted.

■

♢

**Contradiction**

- Assume that the opposite of the hypothesis were true.
- Show that if the opposite were true then the assumptions of the problem would be violated.

This needs more finesse than a proof by induction. There can be a lot more slick when it works.

- Working out small examples of the problem helps.
- Argue about the first/last position where the hypothesis fails to be true.

**Example.** Assume that BUBBLESORT is does not return the best element (smallest) on right. Let $i$ be first position where in the array $s$, $s(i+1) > s(i)$.

1. **Case 1**: $s(i) > s(i+1)$

   Then, BUBBLESORT would have swapped them.

2. **Case 2**: $s(i) < s(i+1)$

   Then, BUBBLESORT would have not swapped them.

♢

# Part II

# Appendices

# BIBLIOGRAPHY