Week 2:

Simple programming problems:

* E.g scanning documents for a work = solution is clear and efficiency is always limited to scanning through every word

Greatest common divisor:

* Puts fraction a/b in its simplest form: a/d/b/d
  + Need d to divide a and b
  + Want d to be as large as possible to reduce the fraction as much as possible
* For integers a and b, the greatest common divisor (GCD) is the best solution
* GCD can be computed using the Euclidean algorithm = optimal algorithm for efficiency:
  + GCD (a,b) = a’ + xb
  + <https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm>

Runtimes:

* Asymptotic runtimes: how does the runtime scale with input size (what are the proportions to n data points)
* If runtime is roughly proportional to n, or if runtime is proportional to n2 or, even slower, 2n
* The relationship between the runtime of the program and the scale of the input is very important in determining efficiency of the algorithm

Big-O Notation:

* Asymptotic notation
* Fn = O(g(n)) (f is big-O of g) or f < g
* E.g 3n2 + 5n + 2 = O(n2) since if n > 1,
* 3n2 + 5n + 2 < 3n2 + 5n +2n2 = 10n2
* This is a method for notating runtime scaling by using a simple notation where constant factors are ignored to simplify the runtimes estimation within a common notation
* 7n3 = O(n3), n2/3 = O(n2)

The main rules for working with logarithms are the following:

1. loga(nk)=klogan
2. loga(nm)=logan+logam
3. nlogab=blogan
4. logan⋅logba=logbn

Recursion:

* For algorithm design, break down a problem into a smaller instance of the problem, and so on, until the problem instance is small enough to solve directly

Common algorithms:

* Greedy algorithm – solve locally optimal problems and aggregate into a global solution
* Divide and conquer – break a large problem into smaller and smaller problems and solve each
* Dynamic programming – solve related problems

Level of design:

Naïve algorithms:

* Slow, and can be definition of the algorithm in the terms of the problem definition

Standard algorithms:

* Apply standard techniques (common algos) to solve the problem

Optimised algorithm:

* Optimise the existing algo to improve efficiency/accuracy

Week 3:

Greedy algorithms:

* Break problems into subproblems
* Maximise the efficiency/value for the problem and solve that subproblem first
* Make a safe move: a move that uses the optimal solution to solve the subproblem
* Iterate

Sorting:

* Sorting can help optimise a greedy algorithm
* E.g sort items by decreasing value, then running time is lower because less combinations have to be considered