1 Week 3

- If $f(n) = O(g(n))$, there exist constants c_1 and c_2 such that $f(n) \leqslant c_2 \cdot g(n)$ holds for all $n \geqslant c_2$.
- If $f(n) = O(g(n))$, we have $\lim_{n\to\infty} \frac{f_1(n)}{g_1(n)} = c$ for some constant c.

2 Week 3 - Extra

- When using 'Direction 1: Constant Finding' setting c_1, always set it to match the coefficent on the LHS so that you can cancel.
- When trying to get a contradiction, try and isolate an $x \cdot c_1$ on the RHS, where $x \in \mathbb{Z}$, such that an expression that contains n is $\leqslant x \cdot c_1$
- Make judicious use of the *max* function when adding functions together
- If $f_1(n) + f_2(n) \leqslant c_1 \cdot g_1(n) + c_1' \cdot g_2(n) \leqslant max\{c_1, c_1'\} \cdot (g_1(n) + g_2(n))$, for all $n \geqslant max\{c_2, c_2'\}$.

3 Week 4

3.1 The Master Theorem

Let $f(n)$ be a function that returns a positive value for every integer $n > 0$. We know:

$$f(1) \leqslant c_1$$
$$f(n) \leqslant \alpha \cdot f(\lceil n/\beta \rceil) + c_2 \cdot n^\gamma \text{ for } n \geqslant 2$$

where $\alpha, \beta, \gamma, c_1$ and c_2 are positive constants. Then:

- If $log_b\alpha < \gamma$ then $f(n) = O(n^\gamma)$
- If $log_b\alpha = \gamma$ then $f(n) = O(n^\gamma \cdot log(n))$
- If $log_b\alpha > \gamma$ then $f(n) = O(n^{log_\beta(a)})$

4 Week 5

TODO

5 RAM Model

5.1 Memory

Infinite sequence of cells, contains w bits. Every cell has an address starting at 1

5.2 CPU

32 registers of width w bits.

5.2.1 Operations

Set value to register (constant or from other register). Take two integers from other registers and store the result of; $a + b$, $a - b$, $a \cdot b$, a/b. Take two registers and compare them; $a < b$, $a = b$, $a > b$. Read and write from memory.

5.3 Definitions

An algorithm is a set of atomic operations. It's cost is is the number of atomic operations. A word is a sequence of w bits

6 Worst-case

Worst-case cost of an algorithm is the longest possible running time of input size n

7 Dictionary search

let n be register 1, and v be register 2
register $left \to 1$, $right \to 1$
while $left \leq right$
  register $mid \to (left + right)/2$
  if the memory cell at address $mid = v$ then
    return yes
  else if memory cell at address $mid > v$ then
    $right = mid - 1$
  else
    $left = mid + 1$
return no

Worst-case time: $f_2(n) = 2 + 6\log_2 n$

8 Big-O

We say that $f(n)$ grows asymptotically no faster than $g(n)$ if there is a constant $c_1 > 0$ such that $f(n) \leq c_1 \cdot g(n)$ and holds for all n at least a constant c_2. This is denoted by $f(n) = O(g(n))$.

8.1 Example

$1000\log_2 n = O(n)$,$n \neq O(10000\log_2 n)$
$\log_{b_1} n = O(\log_{b_2} n)$ for any constants $b_1 > 1$ and $b_2 > 1$. Therefore $f(n) = 2 + 6\log_2 n$ can be represented; $f(n) = O(\log n)$

9 Big-Ω

If $g(n) = O(f(n))$, then $f(n) = \Omega(g(n))$ to indicate that $f(n)$ grows asymptotically no slower than $g(n)$. We say that $f(n)$ grows asymptotically no slower than $g(n)$ if $c_1 > 0$ such $f(n) \geq c_1 \cdot g(n)$ for $n > c_2$; denoted by $f(n) = \Omega(g(n))$

10 Big-Θ

If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then $f(n) = \Theta(g(n))$ to indicate that $f(n)$ grows asymptotically as fast as $g(n)$

11 Sort

11.1 Merge Sort

Divide the array into two parts, sort the individual arrays then combine the arrays together. $f(n) = O(n\log n)$.
This is the fastest sorting time possible (apart from $O(n\log\log n)$

11.2 Counting Sort

A set S of n integers and every integer is in the range [1, U]. (all integers are distinct)
Step 1: Let A be the array storing S. Create array B of length U. Set B to zero.
Step 2: For $i \in [1, n]$; Set x to A[i], Set B[x] = 1
Step 3: Clear A, For $x \in [1, U]$; If B[x] = 0 continue, otherwise append x to A

11.2.1 Analysis

Step 1 and 3 take O(U) time, while Step 2 O(n) time. Therefore running time is O(n + U) = O(U).

12 Random

RANDOM(x, y) returns an integer between x and y chosen uniformly at random

13 A Universal Function

- Pick a prime number p such that
- $p \geqslant m$ and $p \geqslant$ any possible integer k
- Choose a number α uniformly at random from $1, ..., p - 1$
- Choose a number β uniformly at random from $1, ..., p - 1$
- Construct a hash function: $h(k) = 1 + ((\alpha k + \beta) \cdot mod(p)) \cdot mod(m)$

14 SE Set 3

- Find out how many times a recurrence takes to terminate, and then proceed to eyeball the time complexity