

COMP20007 Workshop Week 9

Preparation:

- have draft papers and pen ready, or ready to work on whiteboard
- open [wokshop10.pdf](#) (from [LMS](#)), and
- download lab files from [LMS](#)

1

Hashing: Problems T1, T2

2

Huffman Coding: Problems T3, T4

3

Revision on demands:

Complexity (problem T5)

Solving Recurrences

and others

LAB

Lab: playing with hashing code

- additional: see 16s2.207.w10\
- **DPV5.13.** A long string consists of the four characters A;C; G; T; they appear with frequency 31%; 20%; 9%, and 40%, respectively. What is the Huffman encoding of these four characters?

Exam Question? Informal "Bad" Huffman Algorithm

Input: array $f[n]$ of frequencies

Output: encoding tree

H = empty minHeap, ordered by frequencies

make a node for each $(i, f[i])$ and insert into H

while (H has more than 1 element) :

$\text{deleMin}(H)$ twice, get node $I = (*, f[i])$ and $J = (*, f[j])$

 create new node $K = (-1, f[i] + f[j])$

 make K parent of I and J

 insert K to H

}

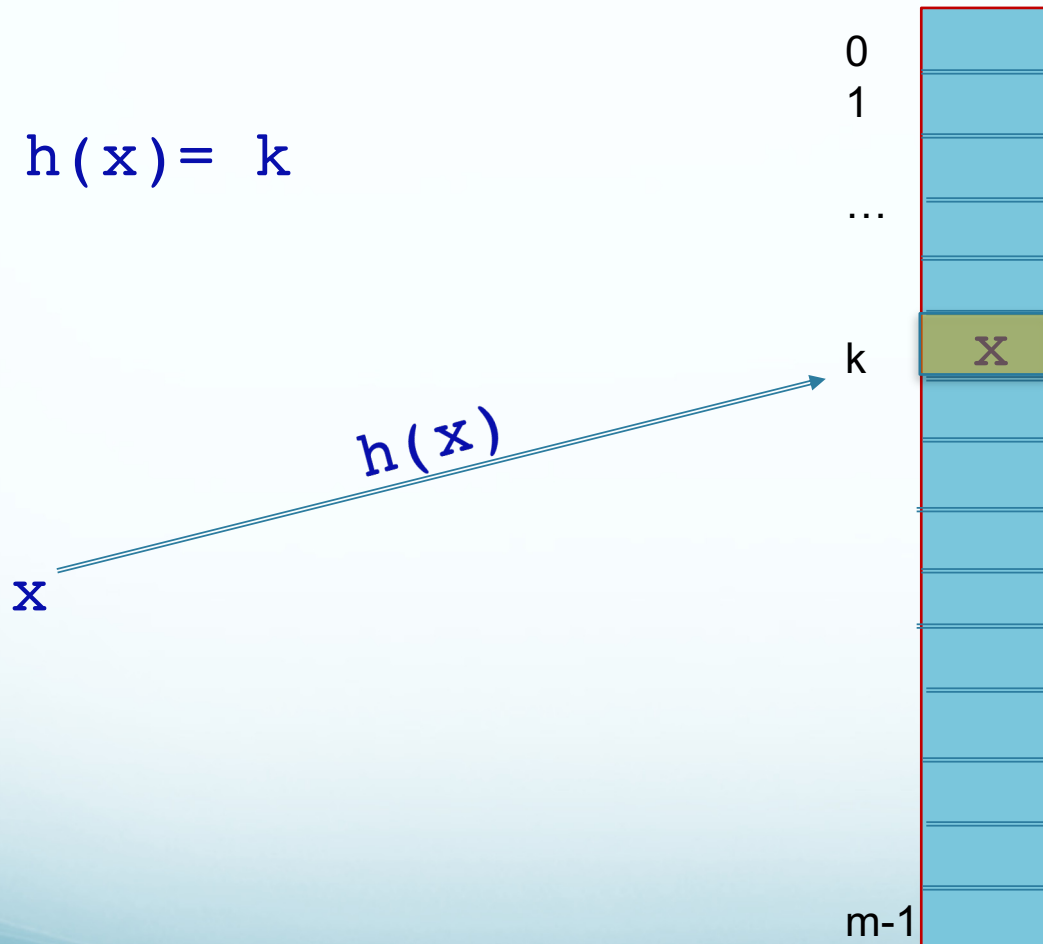
$R = \text{deleMin}(H)$ is the root of the output tree

//note: $*$ means "anything", -1 means "internal node"

Hashing: dictionary with $O(1)$ search/insert

- *Hashing* = hash table $T[m]$ + hash functions $h(x)$: store key x in T , at position $h(x)$

$$h(x) = k$$



Example:
storing a list of
800 student
records, each
student has a
unique student
number in the
range of:

1. 1..999
2. 7001..7999
3. 100..200000

$$h(x) = ?, m = ?$$

Collisions

- $h(x_1) = h(x_2)$ for some $x_1 \neq x_2$.
- Collisions are normally unavoidable.
- Example with student numbers: $m = 997$ (a prime number > 800), $x_1 = 998$, $x_2 = 9971$

Collisions

Example:

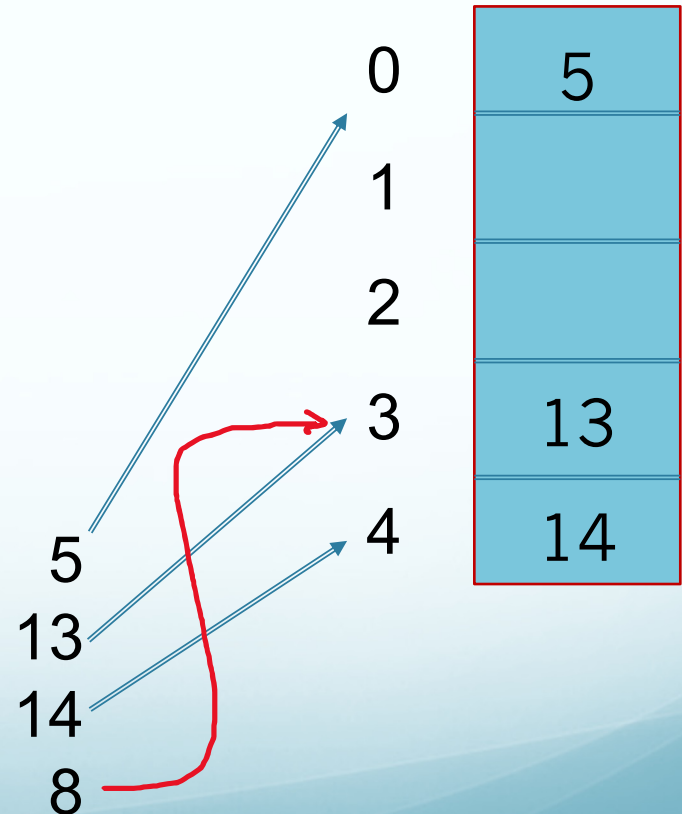
$m=5$, $h(x) = x \% m$

Here: $h(8) = h(5)$

One method *to reduce collisions* using *a prime number* for hash table size m .

Another method is to make the table size m *big* enough (but that affects space efficiency).

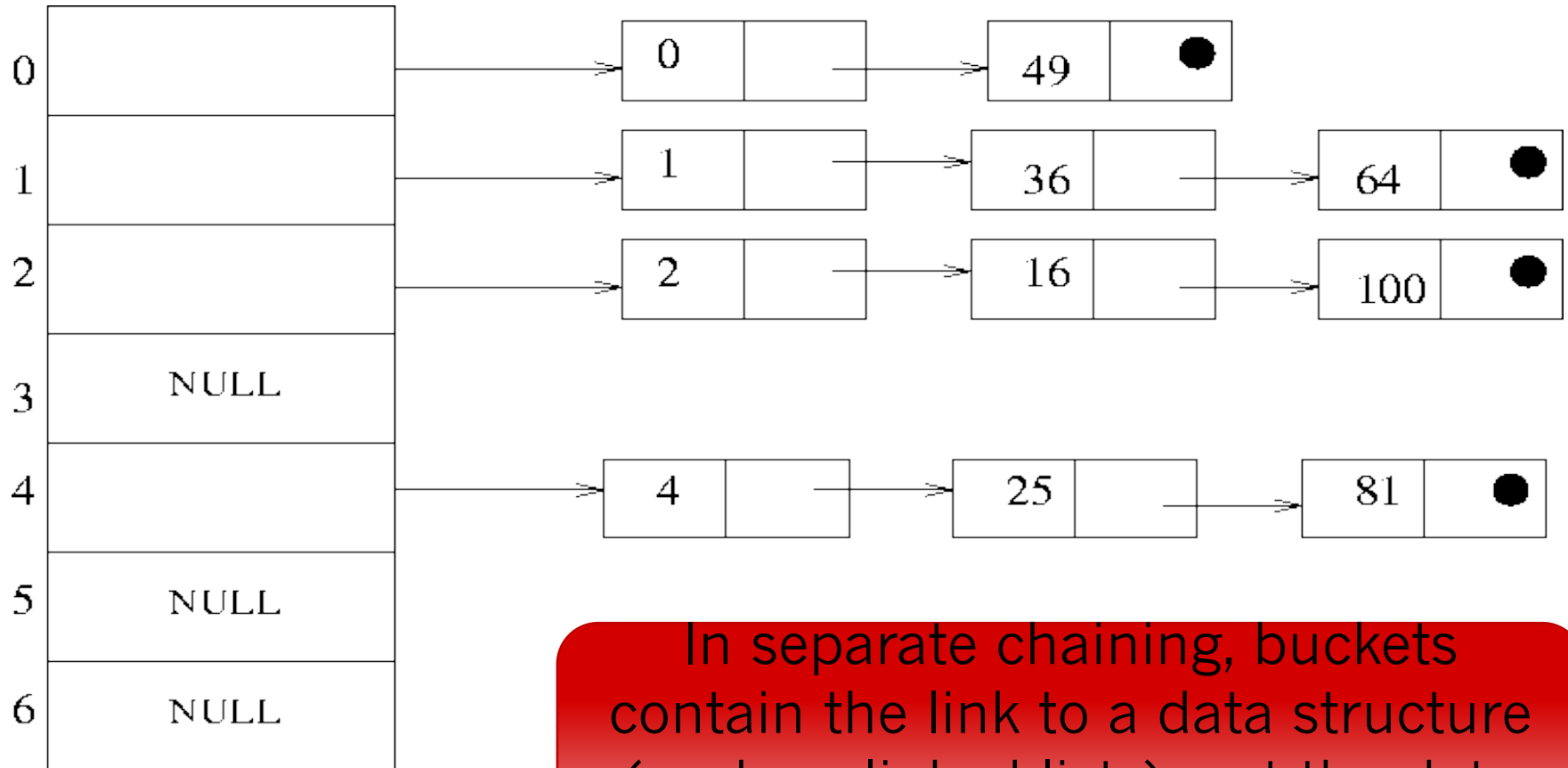
Even though, collisions might still happen



Collision Solution 1: Separate Chaining

$h(x) = x \% 7$, keys entered in decreasing order:

100, 81, 64, 49, 36, 25, 16, 4, 2, 1, 0



In separate chaining, buckets contain the link to a data structure (such as linked lists), not the data themselves.

Solution 2: Linear Probing (here, data are in buckets)

When inserting we do some probes until getting a vacant slot. $H(x, \text{probe})$ can be summarized as:

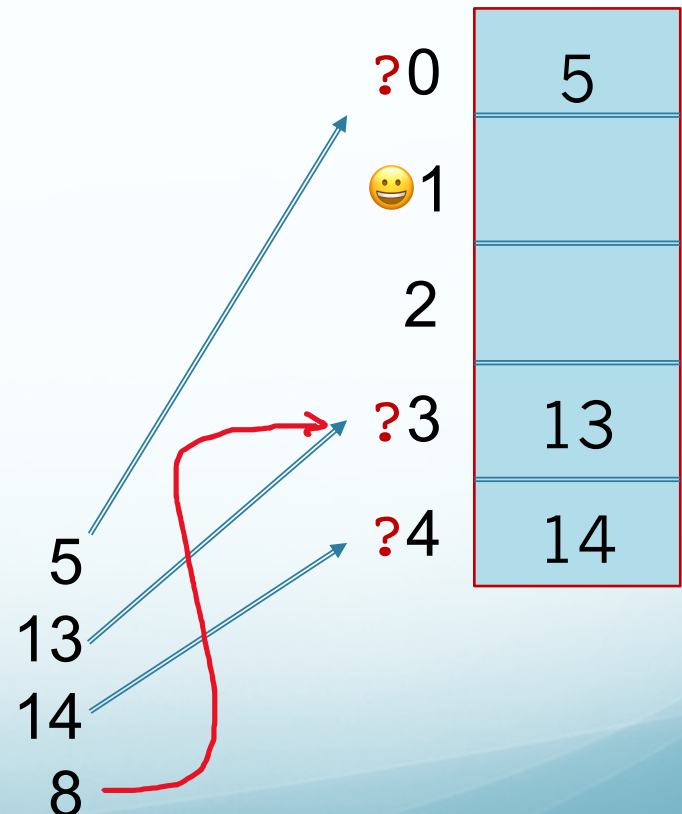
$$h(x) \rightarrow H(x, \text{probe}) = (h(x) + \text{probe}) \bmod m$$

where m probe is $0, 1, 2 \dots$ (until reaching a vacant slot).

Example: $m=5$, $h(x) = x \bmod m$,

Notes:

- The same procedure for search
- Deletion is problematic!



Double hashing

Double hashing is similar to *linear probing*, but employ a second hash function $h_2(x)$:

$$H(x, \text{probe}) = (h(x) + \text{probe} * h_2(x)) \bmod m$$

where probe is $0, 1, 2, \dots$ (until reaching a vacant slot). Note that:

- $h_2(x) \neq 0$ for all x , (why?)
- to be good, $h_2(x)$ should be co-prime with m , (how?)
- *linear probing* is just a special case of *double hashing* when $h_2(x) = 1$.

Problem 1&2 [Group/Individual]: Separate chaining

Problem 1: Consider a hash table in which the elements inserted into each slot are stored in a linked list. The table has a fixed number of slots $L=2$. The hash function to be used is $h(k) = k \bmod L$.

- a) Show the hash table after insertion of records with the keys
17 6 11 21 12 33 5 23 1 8 9
- b) Can you think of a better data structure to use for storing the records that overflow each slot?

Problem 2: Consider a hash table in which each slot can hold one record and additional records are stored elsewhere in the table using linear probing with steps of size $i=1$. The table has a fixed number of slots $L=8$. The hash function to be used is $h(k) = k \bmod L$.

- a) Show the hash table after insertion of records with the keys
17 7 11 33 12 18 9
- b) Repeat using linear probing with steps of size $i = 2$. What problem arises, and what constraints can we place on i and L to prevent it?
- c) Can you think of a better way to find somewhere else in the table to store overflows?

Problem 1 [Group/Individual]: Separate chaining

Consider a hash table in which the elements inserted into each slot are stored in a linked list. The table has a fixed number of slots $L=2$. The hash function to be used is $h(k) = k \bmod L$.

- a) Show the hash table after insertion of records with the keys
17 6 11 21 12 33 5 23 1 8 9
- b) Can you think of a better data structure to use for storing the records that overflow each slot?

Problem 2[Group/Individual]: Open addressing

Consider a hash table in which each slot can hold one record and additional records are stored elsewhere in the table using linear probing with steps of size $i=1$. The table has a fixed number of slots $L=8$. The hash function to be used is $h(k)=k \bmod L$.

- a) Show the hash table after insertion of records with the keys
17 7 11 33 12 18 9
- b) Repeat using linear probing with steps of size $i = 2$. What problem arises, and what constraints can we place on i and L to prevent it?
- c) Can you think of a better way to find somewhere else in the table to store overflows?

Huffman's Coding & Data Compression

The task:

Input: a message T such as `that cat, that bat, that hat` over some alphabet

Output: an encoded message T' – an efficient storage of T with the guarantee that T can be reproduced from T'. For example

T' = 11100011110100010111...

Principle: Use less number of bits (shorter codeword) for symbol that appears more frequently.

Input message: **that cat, that bat, that hat**

alphabet= [**a b c h t ,**] (or perhaps all ASCII characters)

How to compress:

1. Modeling: making assumptions about the structure of messages

th**a**t **c**a**t**, **t**h**a**t **b**a**t**, **t**h**a**t **h**a**t**

(character model)

that **cat,** **that** **bat,** **that** **hat**

(word model)

tha**t** **ca**t**,** **th**a**t** **ba**t**,** **th**a**t** **ha**t

(bi-character model)

2. Statistics: find symbol distribution

3. Coding: build the *code table* and do *encoding*

Input message: **that cat, that bat, that hat**

alphabet= [**a b c h t , _**] (or perhaps all ASCII characters)

1. Modeling: making assumptions about the structure of messages

that cat, that bat, that hat (character model)

2. Statistics: build table of frequencies, aka weight table. For the character model:

a	b	c	h	t	,	_
6/28	1/28	1/28	4/28	9/28	2/28	5/28

or just

a	b	c	h	t	,	_
6	1	1	4	9	2	5

3. Coding: build the *code table* and do *encoding*

a	b	c	h	t	,	_
01	0000	0001	100	11	001	101

→ **11100011110100010111...**

Huffman Coding = a method for building code

Build Huffman code:

- make a node for each weight
- join 2 *smallest weights* and make a parent node (of binary tree), continue until having a single root
- for each node, assign 0- and 1-bit for 2 associated edges

Example:

a	b	c	h	t	,	!
6	1	1	4	9	2	5

Huffman Coding

Note: there are different versions of Huffman code for a same weight table (depending on dealing with ties, assigning 0- and 1-bits), all we need to do is to choose a way and keep consistency. For instance (*canonical Huffman's coding*):

- when joining 2 weights into one, always make the smaller weight be the left child (hence, need to always keeps current roots in weight ordering)
- choose a consistent way for breaking ties
- when assigning code, always set 0 to the left edge, 1 to the right edge

Additional notes

- When sending encoded messages, the sender also need to send the weight table (or something equivalent).
- It's important that the receiver/decoder builds the code in the same way as the sender/coder does.

Problem 3: Huffman Code Generation

Huffman's Algorithm generates prefix-free code trees for a given set of symbol frequencies. Using these algorithms generate two code trees based on the frequencies in the following message:

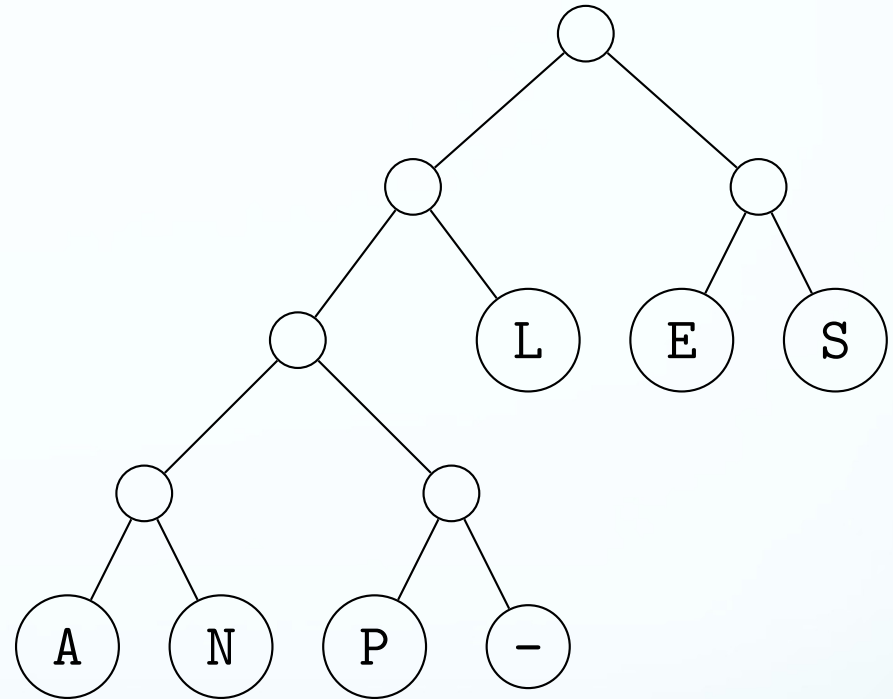
`losslesscodes`

What is the total length of the compressed message using the Huffman code?

Problem 4: Canonical Huffman decoding

The code tree was generated using Huffman's algorithm, and converted into a Canonical Huffman code tree. Note: _ denotes space.

Assign codewords to the symbols in the tree, such that left branches are denoted 0 and right branches are denoted 1. Use the resulting code to decompress the following message:



00100110000011100011011011110011110110100010011011110001101111

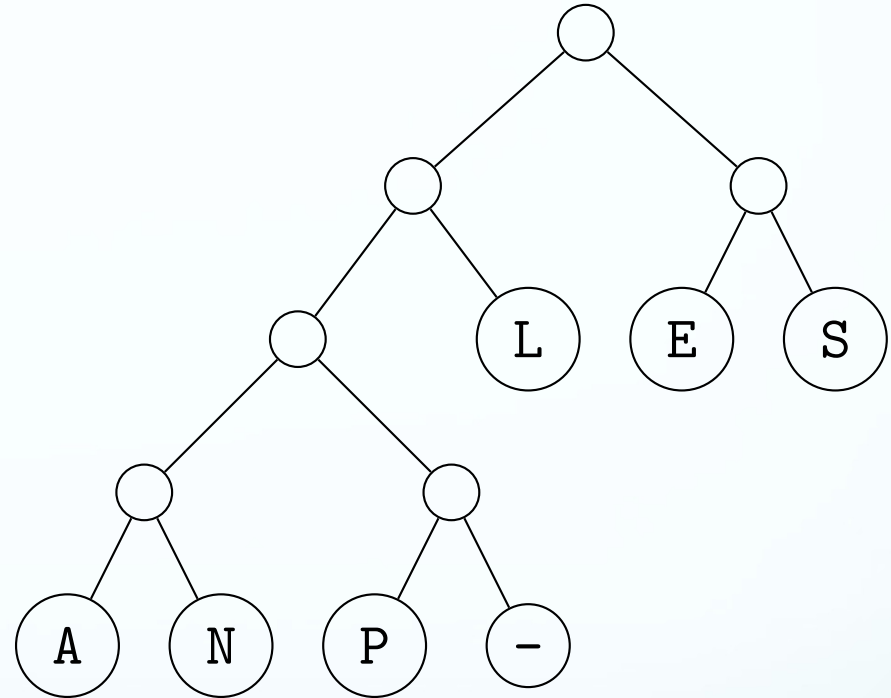
Problem 3&4: Huffman Code Generation

Problem 3: Huffman's Algorithm generates prefix-free code trees for a given set of symbol frequencies. Using these algorithms generate two code trees based on the frequencies in the following message:

losslesscodes

What is the total length of the compressed message using the Huffman code?

Problem 4: the code tree



Problem 4: Decode:

00100110000011100011011011110011110110100010011011110001101111

Revision 1: Complexity Analysis – 03.pdf & 04.pdf

$$1 < \log n < n^\epsilon < n^c < n^{\log n} < c^n < n^n \quad \text{where } 0 < \epsilon < 1 < c$$

$$\begin{aligned} O(f(n) + g(n)) &= O(\max\{f(n), g(n)\}) \\ O(cf(n)) &= O(f(n)) \\ O(f(n) \times g(n)) &= O(f(n)) \times O(g(n)) \end{aligned} \quad \begin{array}{l} \text{note: these 3 also applied} \\ \text{to big-}\theta \end{array}$$

$$\begin{aligned} 1+2+\dots+n &= n(n+1)/2 &= \theta(n^2) \\ 1^2+2^2+\dots+n^2 &= n(n+1)(2n+1)/6 &= \theta(n^3) \\ 1+x+x^2+\dots+x^n &= (x^{n+1}-1)/(x-1) \quad (x \neq 1) \end{aligned}$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f(n) = O(g(n)) \\ c & f(n) = \theta(g(n)) \\ \infty & f(n) = \Omega(g(n)) \end{cases}$$

$$\lim_{n \rightarrow \infty} \frac{t(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{t'(n)}{g'(n)}$$

Revision exercises: Problem 5

For each of the following cases, indicate whether $f(n)$ is $O(g(n))$, or $\Omega(g(n))$, or both (that is, $\Theta(g(n))$)

(a) $f(n) = (n^3 + 1)^6$ and $g(n) = (n^6 + 1)^3$,

(b) $f(n) = 3^{3n}$ and $g(n) = 3^{2n}$,

(c) $f(n) = \sqrt{n}$ and $g(n) = 10n^{0.4}$,

(d) $f(n) = 2 \log_2\{(n + 50)^5\}$ and $g(n) = (\log_e(n))^3$,

(e) $f(n) = (n^2 + 3)!$ and $g(n) = (2n + 3)!$,

(f) $f(n) = \sqrt{n^5}$ and $g(n) = n^3 + 20n^2$.

Other exercises: review exercises and solution for Workshop Week 3,

Lab

Download [lab_files.zip](#), unzip it, and “play” with the hashing code by following the instructions in the [workshop10.pdf](#) sheet.

and/or continue with reviewing.