Final Exam: CMPT231

Tue 13 Dec 2016 14:00-16:00

Exam Policies

• **Allowed**:
  ◦ Paper copy of **textbook**
  ◦ Paper **notes**, printed or handwritten
  ◦ Bring **blank** 8.5x11" paper
	• **Not** allowed:
	  ◦ Use of **electronics**, computers, etc.
	  ◦ Use of mobile **phone** (in pocket/bag)
	  ◦ **Communication** with anyone except invigilator

- Consider vertices in **alphabetical** order, and edges in **lexicographic** order of endpoints.
- Please **show** as much work as you can.
- If you feel a question is **ambiguous**, explain your interpretation and answer accordingly.
- Please label/**number** your pages clearly and **staple** them in order when handing in.

Final exam [110 pts; counted out of 100]

1. (3 *pts*) Say you are sorting an array that is nearly **pre-sorted**: e.g., only **one** element is out of place. **Which** sorting algorithm (of the ones we learned) would be most appropriate? **Why**?

2. (3 *pts*) In a table of millions of users, each user has a unique 32-bit **ID number**. You wish to sort by ID number. **Which** sorting algorithm (of the ones we learned) would be most appropriate? **Why**?

3. (6 *pts*) **Prove** from definition: $10n^2 \log n + O\Big(n \log^2 n\Big) = O\big(n^2 \log n\big)$

4. (6 *pts*) Let $F_0 = F_1 = 1, \text{ and } F_n = F_{n-2} + F_{n-1}, \forall n > 1$.
   Let $\phi = \dfrac{1+\sqrt{5}}{2} \approx 1.62$, such that $\phi^2 = \phi + 1$. **Prove by induction**: $F_n \leq \phi^n$

5. (4 *pts*) Solve the **recurrence**: $T(n) = 4T\Big(\dfrac{n}{2}\Big) + n^2 \log_2^2 n$

6. (8 *pts*) Demonstrate **heap sort** on the following. How many **swaps**? `[ F, L, B, C, M, X, D, V, T, S ]`

7. (4 *pts*) Suppose we have a **partitioning** scheme, called `LopsidedPart`, for Quicksort. `LopsidedPart` can perform a partition in $O(n)$ time, but the larger portion can be up to 99% of the array. What is the **worst-case** complexity of Quicksort using `LopsidedPart`? Why?

8. (4 *pts*) **Prove** or disprove: **counting sort** on a list of n integers runs in $O(n)$ time.

9. (6 *pts*) Is **deletion** in a binary search tree **commutative**? I.e., do "del(x), del(y)" and "del(y), del(x)" always yield the same tree? **Prove** or give a smallest counterexample. Assume deletion uses successor.

10. (4 *pts*) Recall that **depth-first search** is $\Theta(V + E)$ when using adjacency lists.
    What is its running time if an **adjacency matrix** is used instead?

11. Demonstrate inserting the following in order. If insertion fails, just note it.
    `[ 10, 15, 12, 14, 1, 13, 4, 23, 7, 9 ]`
    a. (3 *pts*) A hash table of size $m = 11$ with division hash and **chaining**
    b. (3 *pts*) A hash table of size $m = 11$ with division hash and **linear probing**
    c. (3 *pts*) A hash table of size $m = 11$ with division hash and **quadratic probing**: $c_1 = 0, c_2 = 1$
    d. (4 *pts*) Same, with **double-hashing**: h_1 = division hash, and $h_2(k) = 5 - (k \bmod 5)$
    e. (3 *pts*) A **binary search tree**
    f. (6 *pts*) A **B-tree** with $t = 2$ (using pre-emptive split/merge, as in lecture)
    g. (2 *pts*) Now, **delete** 13 from the B-tree.

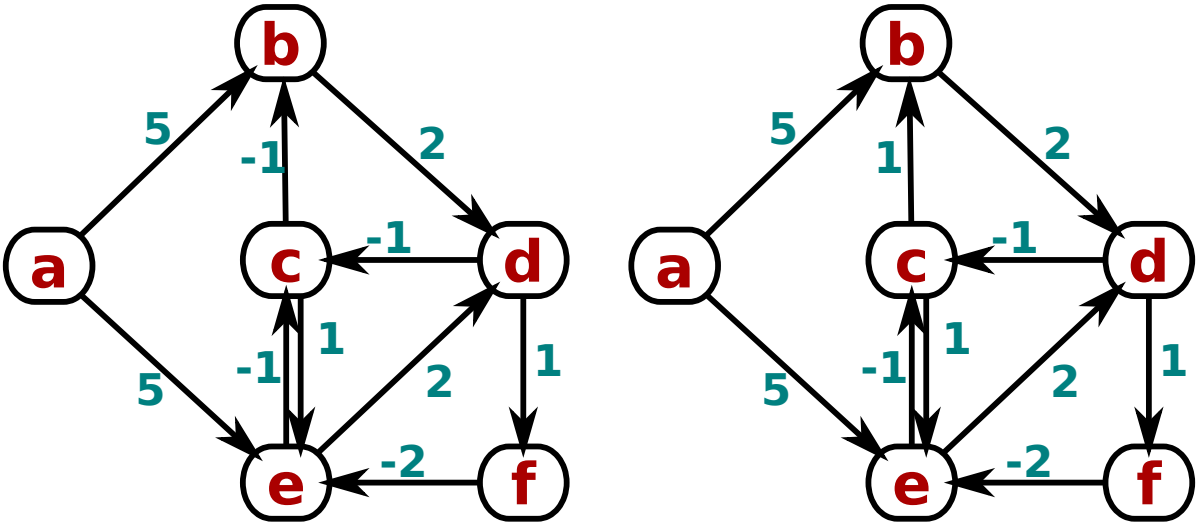
12. You are managing a large youth-sports database; each participant is assigned one of nine **team colours**. The list of colour assignments is very long, and you would like to **compress** it. The colours and relative frequencies are given below:

R	O	Y	G	B	I	V	W	K
28%	6%	4%	16%	26%	2%	8%	3%	7%

    a. (6 *pts*) Construct a **Huffman tree**, following the pseudocode in lecture.
    b. (2 *pts*) **Encode** each colour in binary using your Huffman tree.
    c. (2 *pts*) Derive the **compression ratio** relative to fixed-length encoding.

13. You are a wedding planner, deciding the guests' **seating arrangement**. For each guest u, you have a list of which other guests v are **known** by u. Assume **commutativity**; i.e., if u knows v, then v also knows u. We wish to **minimise** the number of tables while ensuring that each guest knows **every** other guest at his/her table, either directly or indirectly (via a series of intermediate friends).
    a. (5 *pts*) **Describe** an efficient algorithm to solve this.
    b. (2 *pts*) Analyse the **complexity** of your algorithm.

14. (9 *pts*) In the **left-hand** graph below, demonstrate **Bellman-Ford** for shortest paths from the source a:



15. (9 *pts*) The **right-hand** graph above is the same except $w(c, b) = 1$. Demonstrate **Floyd-Warshall** for all-pairs shortest paths on the right-hand graph. Show the D matrix after each iteration k.

16. (3 *pts*) Which of the many algorithms we learned in this course is your **favourite**, or sticks in your mind the most? **Why**?