COMS20010 — Problem sheet 6

You don't have to finish the problem sheets before the class — focus on understanding the material the problem sheet is based on. If working on the problem sheet is the best way of doing that, for you, then that's what you should do, but don't be afraid to spend the time going back over the quiz and videos instead. (Then come back to the sheet when you need to revise for the exam!) I'll make solutions available shortly after the problem class. As with the Blackboard quizzes, question difficulty is denoted as follows:

- ⋆ You'll need to understand facts from the lecture notes.
- ⋆⋆ You'll need to understand and apply facts and methods from the lecture notes in unfamiliar situations.
- ⋆⋆⋆ You'll need to understand and apply facts and methods from the lecture notes and also move a bit beyond them, e.g. by seeing how to modify an algorithm.
- ⋆⋆⋆⋆ You'll need to understand and apply facts and methods from the lecture notes in a way that requires significant creativity. You should expect to spend at least 5–10 minutes thinking about the question before you see how to answer it, and maybe far longer. Only 20% of marks in the exam will be from questions set at this level.
- ⋆⋆⋆⋆⋆ These questions are harder than anything that will appear on the exam, and are intended for the strongest students to test themselves. It's worth trying them if you're interested in doing an algorithms-based project next year — whether you manage them or not, if you enjoy thinking about them then it would be a good fit.

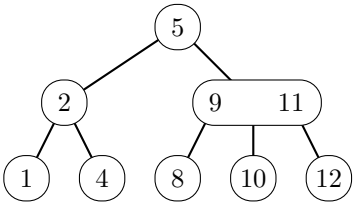
If you think there is an error in the sheet, please post to the unit Team — if you're right then it's much better for everyone to know about the problem, and if you're wrong then there are probably other people confused about the same thing.

This problem sheet covers week 6, focusing on spanning trees, the union-find data structure and 2-3-4 trees.

1. (a) [⋆⋆] Consider the list of numbers $5, 2, 7, 9, 8, 11, 13, 4, 6, 14$. Add these to a 2-3-4 tree in this order and draw what the final tree looks like.

   (b) [⋆⋆] Now remove the elements $2, 9, 4$. What does the final tree look like?

   (c) [⋆⋆] Consider the 2-3-4 tree

   What does this tree look like after running `Delete(4)`?

2. [⋆⋆] In database design, it is very common for each record to have a unique identifier, and for records to be added and removed while the database is running. Explain how 2-3-4 trees could be used to support a database.

3. [⋆⋆] Prove, using the perfect balance property, that a 2-3-4 tree containing n vertices has depth $\Theta(\log n)$.

4. (a) [⋆⋆] Consider the list $8, 3, 16, 1, 6, 19, 2$. Add each of these to a union-find data structure (i.e. run `MakeUnionFind`). What does the data structure look like after running this operation?

(b) [⋆⋆] Run `Union(8,1)`, `Union(19,16)`, `Union(2,16)`, `Union(1,3)`, `Union(16,8)`. What does the data structure look like after running these operations?

(c) [⋆⋆] What is the result of running `FindSet(1)`, `FindSet(2)`, `FindSet(19)`?

5. [⋆⋆⋆] Let G be an m-edge connected weighted graph, given in adjacency list form, whose edge weights are all either 1 or 2. Adapt Kruskal's algorithm into an $O(m\alpha(m))$-time algorithm to find a minimum spanning tree of G, where α is the inverse Ackermann function.

6. [⋆⋆] Let T be a 2-3-4 tree.

   (a) Prove by induction that performing an in-order traversal of T will output the values of T in increasing order. (For example, on encountering a 3-node with values 5 and 7, and children c_0, c_1 and c_2 from left to right, in-order traversal first recursively processes c_0, then outputs 5, then processes c_1, then outputs 7, then processes c_2.)

   (b) Prove that if v is a value stored in a non-leaf node N of T, then the predecessor of v is stored in a leaf.

7. [⋆⋆⋆] Let G be a connected weighted graph, and suppose that no two edges in G have the same weight. Prove that G has a **unique** minimum spanning tree. (**Hint:** Look back at the correctness proof for Kruskal's algorithm.)

8. (a) [⋆⋆] Let T_1 and T_2 be 2-3-4 trees, containing n_1 and n_2 values respectively, where $n_1 \leq n_2$ and the depth of T_1 is at most the depth of T_2. Give an algorithm to merge them into a single tree in $O(n_1 \log(n_2))$ time. (You may assume all values in T_1 and T_2 are distinct from each other.)

   (b) [⋆⋆] Explain intuitively why we should expect that any algorithm to do this will require $\Omega(n_1)$ time.

   (c) [⋆⋆⋆⋆⋆] Suppose now that we are given a single element x, that every value in T_1 is strictly less than x, and that every value in T_2 is strictly greater than x. (This operation is called a *join*.) Give an algorithm to merge T_1, T_2 and x into a single 2-3-4 tree in $O(\log n_2)$ time. You do not need to prove it works.

   (d) [⋆⋆⋆] Now give an algorithm to merge only T_1 and T_2 into a single 2-3-4 tree in $O(\log n_2)$ time. (**Hint:** Try reducing to part (c).)

9. [⋆⋆⋆⋆] You are given an $n \times n$ bitmap image, and you wish to divide the pixels into contiguous regions of similar colours. (This is a common problem in computer vision.) We model this by a grid graph G with vertex set $[n] \times [n]$. We say a grid square (i, j) is adjacent to the grid square to its left, right, top or bottom if their corresponding pixels have similar RGB values. Suppose your computer has $C \in O(n)$ cores working in parallel, all of which have read and write access to a common union-find data structure initialised with all points in $[n] \times [n]$. Sketch an algorithm to find the components of G in $O(n^2\alpha(n^2)/C)$ time.