




Algorithms and Analysis

COSC 2123/1285

Assignment 2: Algorithm Design & Complexity Analysis

	Assessment Type	Individual Assignment. Submit online via Canvas → Assignments → Assignment 2. Clarifications/updates/FAQs can be found in Ed Forum → Assignment 2: General Discussions.
	Due Date	Week 12, 11:59pm, May 27, 2022
	Marks	40

IMPORTANT NOTES

- **If you are asked to design an algorithm**, you need to describe it in plain English first, say a paragraph, and then provide an **unambiguous** pseudo code, unless specified otherwise. The description must include enough details to understand how the algorithm runs and what the complexity is roughly. All algorithm descriptions and pseudo codes required in this assignment are at most half a page in length. **Worst-case complexity** is assumed unless specified otherwise.
- Standard array operations such as sorting, linear search, binary search, sum, max/min elements, as well as algorithms discussed in the pre-recorded lectures can be used straight away (but make sure to include the input and output if you are using them as a library). However, if some modification is needed, you have to provide a full description. If you are not clear whether certain algorithms/operations are standard or not, post it to Ed Discussion Forum or drop Hoang a Team message.
- Marks are given based on **correctness**, **conciseness** (with page limits), and **clarity** of your answers. If the marker thinks that the answer is completely not understandable, a zero mark might be given. If correct, ambiguous solutions may still receive a deduction of 0.5 mark for the lack of clarity.
- **Page limits** apply to ALL problems in this assignment. Over-length answers may attract mark deduction (0.5 per question). We do this to (1) make sure you develop a concise solution and (2) to keep the reading/marking time under control. **Please do NOT include the problem statements in your submission** because this may increase Turnitin's similarity scores significantly.
- This is an individual assignment. While you are encouraged to seek clarifications for questions on Ed Forum, please do NOT discuss solutions or post hints leading to solutions.

- In the submission (your PDF file), you will be required to certify that the submitted solution represents your own work only by agreeing to the following statement:

I certify that this is all my own original work. If I took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in my submission. I will show that I agree to this honour code by typing "Yes":

1 Part I: Fundamental

Problem 1 (8 marks, 1 page). Consider the algorithm **mystery()** whose input is a binary tree, or more precisely, its root R . We denote by R_{Left} and R_{Right} the left and the right children of R in the tree.

Algorithm **mystery**(R : root of a binary tree)

```
if  $R = \emptyset$  then
    return 0;
else
    return  $1 + \text{mystery}(R_{\text{Left}}) + \text{mystery}(R_{\text{Right}})$ ;
end if
```

- a) [2 marks] What does the algorithm compute? Justify your answer.
- b) [1 mark] What is the algorithmic paradigm that the algorithm belongs to?
- c) [2 marks] Assume that the tree is a perfect binary tree of height h (a binary tree is called perfect if every non-leaf node has precisely two children and all leaf-nodes are at the same level). Write the recurrence relation for $C(h)$, the number of **additions** required by **mystery()**. Convention: a single-node tree has height 0.
- d) [2 marks] Solve the above recurrence relation by the backward substitution method to obtain an explicit formula for $C(h)$ in h for the perfect binary tree of height h .
- e) [1 mark] Write the complexity class that $C(h)$ belongs to using the Big- Θ notation.

Problem 2 (8 marks, 1.5 pages). The Australian Government Department of Health maintains a list of n people who have been double vaccinated against Covid, called `list_double`. At the end of 2022, an aggregated list of m people who have been vaccinated the third time in 2022 with booster shots is created, called `list_triple`. Note that $m \leq n$. People in each list are identified by their unique ID numbers (e.g., passport numbers) and are entered into the lists in chronological order. It is required to design an algorithm that takes as input the two lists and returns a new list of people who are double vaccinated but haven't received their third shots. A reminder will be sent to all people in this list to take their booster shots.

- a) [2 marks] Design (describe + complexity analysis) a brute-force algorithm that performs the aforementioned task [1 mark]. Analyse the time complexity of the algorithm using the big-O notation [1 mark]. Pseudocode is NOT required.
- b) [3 marks] Design (describe + complexity analysis) a transform-and-conquer algorithm with time complexity $O(n \log m)$ that performs the aforementioned task using at most a constant amount of extra space (apart from the input/output).
- c) [3 marks] Design (describe + pseudocode + complexity analysis) an algorithm with (average-case) time complexity $O(n)$ that performs the aforementioned task [2 marks]. There is NO restriction on the space complexity.

Problem 3. [10 marks, 1.5 pages] (**Dijkstra's algorithm + min-heap**) Given a graph as in Fig. 1, we are interested in finding the **shortest paths** from the source a to all other vertices using the Dijkstra's algorithm and a min-heap as a priority queue. Note that a min-heap is the same as a max-heap, except that the key stored at a parent node is required to be smaller than or equal to the keys stored at its two child nodes. In the context of the Dijkstra's algorithm, a node in the min-heap tree has the format $v(p_v, d_v)$, where d_v is the length of the current shortest path from the source to v and p_v is the second to last node along that part (right before v). For example, $b(a, 4)$ is one such node. We treat d_v as the key of Node v in the heap, where $v \in \{a, b, c, d, e, f, g, h\}$.

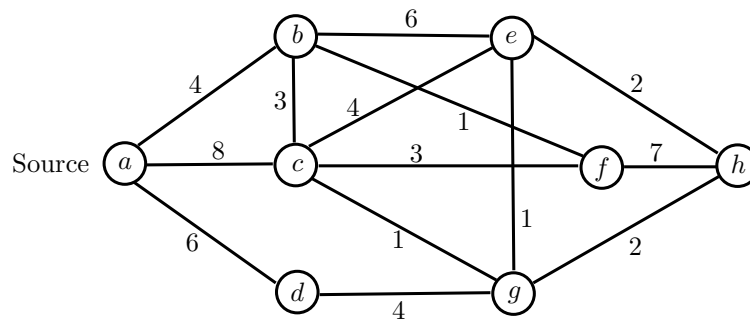


Figure 1: An input graph for the Dijkstra's algorithm. Edge weights are given as integers next to the edges. For example, the weight of the edge (a, b) is 4.

- a) [1 mark] The min-heap after $a(a, 0)$ is removed is given in Fig. 2. The next node to be removed from the heap is $b(a, 4)$. **Draw the heap** after $b(a, 4)$ has been removed and the heap has been heapified (fixed), assuming that $\infty \geq \infty$. No intermediate steps are required.

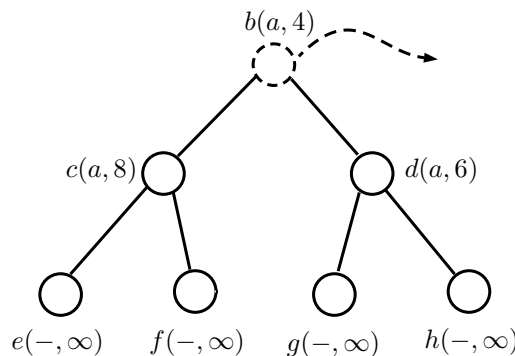


Figure 2: The min-heap (priority queue) after $a(a, 0)$ has been removed.

- b) [2 marks] **Draw the heap(s)** after the neighbours of b have been updated and the heap has been heapified (see the pseudocode in the lecture Slide 30, Week 9). If there are multiple updates then draw multiple heaps, each of which is obtained after one update. Note that neighbours are updated in the alphabetical order, e.g., b must be updated before c . No intermediate steps are required. Follow the discussion on Ed Forum for how to update a node in a heap.

	S : vertices whose shortest paths have been known	Priority queue of remaining vertices
1	$a(a, 0)$	$b(a, 4), c(a, 8), d(a, 6), e(-, \infty), f(-, \infty), g(-, \infty), h(-, \infty)$
2	$a(a, 0), b(a, 4)$	
3		
4		
5		
6		
7		
8		

Table 1: Complete this table for Part c).

- c) [5 marks] Complete Table 1 with correct answers. You are required to follow strictly the steps in the Dijkstra's algorithm taught in the lecture of Week 9.
- d) [2 marks] Fill Table 2 with the **shortest paths** AND the corresponding **distances** from a to ALL other vertices in the format $a \rightarrow ? \rightarrow ? \rightarrow v \mid d_v$, for instance, $a \rightarrow b \mid 4$.

	Shortest Paths	Distances
a	$a \rightarrow a$	0
b	$a \rightarrow b$	4
c		
d		
e		
f		
g		
h		

Table 2: Complete this table for Part d).

2 Part II: Advanced

Problem 4 (7 marks, 2 pages). Each Bitcoin transaction, in its simplest form, has one input coin and several output coins (see Fig. 3). The input coin is *genuine* in the sense that it is spent by the transaction. This creates the issue of traceability for Bitcoin, that is, the entire history of each coin can be traced, e.g., how it was created, split, spent, and by which users, etc., which can sometimes be too revealing and undesirable for the users. To make the cryptocurrency untraceable, it has been proposed that instead of using only genuine inputs, the cryptocurrency wallet should also include other *fake* inputs so that an observer won't know which input is the genuine one for that transaction. We will ignore the fact how this can be done technically and only focus on the mixing part where genuine and fake inputs are mixed in the transactions to provide untraceability. Assume that each coin can be used as a genuine input for exactly one transaction and that the number of coins is the same as the number of transactions in the system.

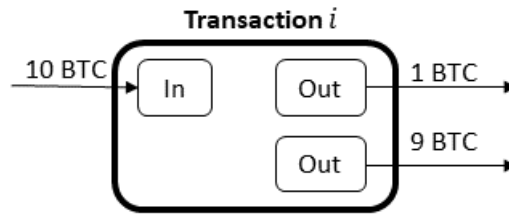


Figure 3: Example of input and output coins in a Bitcoin transaction.

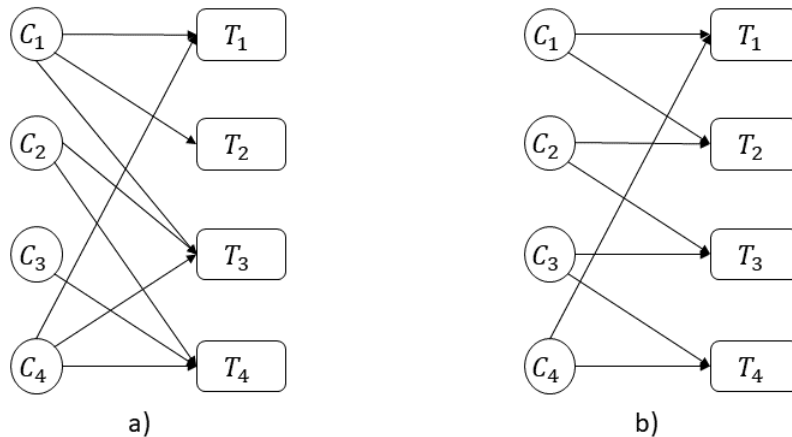


Figure 4: Examples of mixing genuine and fake inputs for transactions to provide untraceability. The mixing strategy in a) fails because an observer can determine a unique mapping M that matches genuine coins with transactions: $M[1] = 2$, $M[2] = 3$, $M[3] = 4$, and $M[4] = 1$. The mixing in b) is not the best in disguising the actual mapping, but still confuses an observer as there are *two* possibilities to match the coins with the transactions.

The mixing strategy sometimes fails because not all mixings are done in a proper way. For example, in Fig. 4 a), an observer can determine which coin is the genuine input of which transaction for ALL the coins. We refer to such a mixing as a *bad* mixing strategy.

Note that a mixing strategy can be described by a bipartite graph with the left-side vertices corresponding to the coins and the right-side vertices corresponding to the transactions, and there is an edge between a coin C_i and a transaction T_j if C_i is included in T_j as an input (genuine or fake). Design an algorithm of time complexity $O(n+m)$, where n is the number of coins (or equivalently, the number of transactions) and m is the number of edges in the bipartite graph, that determines if a particular mixing is bad, i.e., a unique mapping M that maps ALL coins to their corresponding transactions could be found. The algorithm must output M if the mixing is bad.

- a) [3 marks] **Describe** the algorithm in plain English together with a short **pseudocode**. The algorithm must be described in an unambiguous and concise way and provides enough details so that another student can understand how it works and why it solves the problem. The **input** of the algorithm is n , m , the (adjacency) lists of transactions L_i (abbreviation for “Left”) that include the coin C_i as an input, $i = 1, 2, \dots, n$, and the (adjacency) lists of coins R_j (abbreviation for “Right”) that are inputs of Transaction T_j , $j = 1, 2, \dots, n$. The **output** of the algorithm is either the unique mapping M of coins and transactions or None, which indicates that an unique mapping can’t be found, i.e., there are more than one valid mappings.
- c) [2 marks] Demonstrate your algorithm with an example, e.g., in Fig. 4 a).
- d) [2 marks] Show that the complexity is indeed in $O(n+m)$, which means that there are constants a and b such that the complexity is at most $a \times n + b \times m$, e.g. $3n + 2m$.

Problem 5 (7 marks). The n nodes of a rooted complete binary tree can be indexed by the set $[0, n-1] \triangleq \{0, 1, \dots, n-1\}$ in the natural order (top to bottom, left to right). A labelling or permutation p is a map that assigns to each vertex $i \in [0, n-1]$ a label $p[i] \in [0, n-1]$ so that different nodes have different labels. A labelling p is called **magical** if it satisfies the following condition: if we assign to each edge (i, j) of the tree the label $|p[i] - p[j]|$, i.e., the absolute value of the difference between the labels of Node i and Node j , then all edge labels must all be distinct. We can also treat p as a permutation of the vertice labels. See Fig. 5 for an example of such labellings/permutations on trees of $n = 5$ and $n = 8$ nodes. In these cases, the labellings are $p = [0, 4, 2, 1, 3]$ and $p = [3, 0, 5, 7, 6, 4, 1, 2]$. Note that for the first labelling, $p[0] = 0, p[1] = 4, p[2] = 2, p[3] = 1, p[4] = 3$.

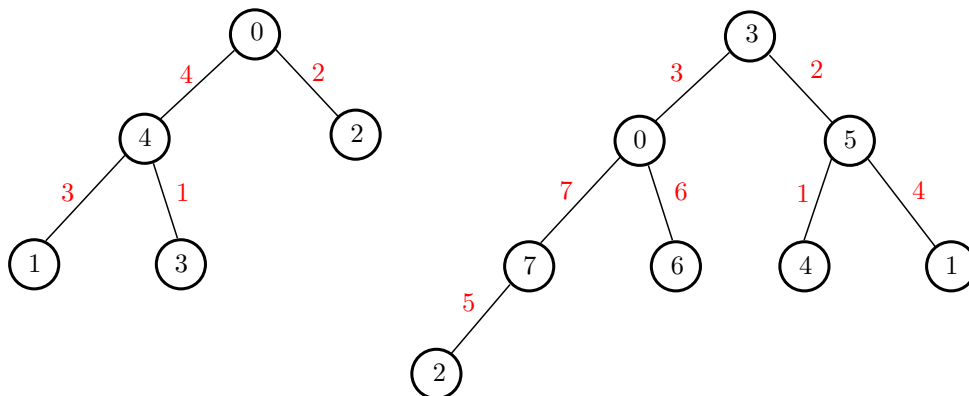


Figure 5: Examples of **magical labellings** $p = [0, 4, 2, 1, 3]$ and $p = [3, 0, 5, 7, 6, 4, 1, 2]$ for complete binary trees of $n = 5$ and $n = 8$ nodes, respectively. The edge labels (in red) are calculated by taking the absolute values of the differences between the labels of two incident nodes, e.g., for the 5-node tree, the edge connecting the root and its left child has label $|p[0] - p[1]| = |0 - 4| = 4$. Both labellings are magical because all edge labels are distinct.

- a) [7 marks, 1.5 pages] Design an efficient algorithm (algorithm description, pseudo code, examples, Python code, and an estimated time complexity if possible) that finds a magical labelling p for every rooted complete binary tree of n nodes. The algorithm is deemed efficient if the submitted implementation in Python can find magical labellings for ALL $1 \leq n \leq 19$ in less than 5 minutes on a laptop. All 19 output magical labellings/permutations must be included in file `permutations.txt` submitted along the code. A Python code to test these permutations is available on [Assignment 2: General Discussion](#) (Ed Discussion Forum).
- b) [1 bonus mark, 2 pages] Provide a mathematical proof that the algorithm developed in Part a) can find a magical labelling for complete binary of n nodes for ALL $n \geq 1$.

Note that there are no partial marks given to this problem. You will receive marks for the problem only if your submitted code can produce magical labellings for ALL $n \leq 19$ in less than 5 minutes AND your description of the algorithm is clear (with examples), correct, and understandable. More details on how to prepare the submission for Problem 5 will be updated on [Assignment 2: General Discussion](#) (Ed Discussion Forum).

3 Submission

The final submission (via Canvas) will consist of:

- Your solutions to all questions in a PDF file of font size 12pt and your agreement to the honour code (see the first page). You may also submit the code in Problem 5.

Late Submission Penalty: Late submissions will incur a 10% penalty on the total marks of the corresponding assessment task per day or part of day late, i.e, 4 marks per day. Submissions that are late by 5 days or more are not accepted and will be awarded zero, unless Special Consideration has been granted. Granted Special Considerations with new due date set after the results have been released (typically 2 weeks after the deadline) will automatically result in **an equivalent assessment in the form of a practical test**, assessing the same knowledge and skills of the assignment (location and time to be arranged by the coordinator). Please ensure your submission is correct and up-to-date, re-submissions after the due date and time will be considered as late submissions. The core teaching servers and Canvas can be slow, so please do double check ensure you have your assignments done and submitted a little before the submission deadline to avoid submitting late.

Assessment declaration: By submitting this assessment, you agree to the assessment declaration - <https://www.rmit.edu.au/students/student-essentials/assessment-and-exams/assessment/assessment-declaration>

4 Plagiarism Policy

University Policy on Academic Honesty and Plagiarism: You are reminded that all submitted work in this subject is to be the work of you alone. It should not be shared with other students. Multiple automated similarity checking software will be used to compare submissions. It is University policy that cheating by students in any form is not permitted, and that work submitted for assessment purposes must be the independent work of the student(s) concerned. Plagiarism of any form will result in zero marks being given for this assessment, and can result in disciplinary action.

For more details, please see the policy at <https://www.rmit.edu.au/students/student-essentials/assessment-and-results/academic-integrity>.

5 Getting Help

There are multiple venues to get help. We will hold separate Q&A sessions exclusively for Assignment 2. We encourage you to check and participate in the Ed Discussion Forum, on which we have a pinned discussion thread for this assignment. Although we encourage participation in the forums, please refrain from posting solutions or suggestions leading to solutions.