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 Canvas Announcements and Discussion → Assignment 2 Queries.
	Due Date	Week 12, 11:59pm, October 10, 2021
	Marks	40

IMPORTANT NOTES

- **If you are asked to develop/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.

- 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 Canvas Discussion Forum or drop us an email at sonhoang.dau@rmit.edu.au.

- 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.

- All stories are fictitious and just for fun. Please do not take them seriously.

- 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). **Saving the niece.**

One day, your niece comes to visit you with a worried look on her face. It turns out that her teacher just gave a very tricky problem as part of their VCE Algorithmics Unit 3. She has spent 3 days working on it without any success. As a loving (and very capable) uncle/aunt, you must help her out. Here is the problem.

Consider the algorithm **mystery()** whose input is an integer array A of size n.

```
Algorithm mystery($A[0\dots(n-1)]$)
  return mysteryRecursive($A[0\dots(n-1)]$);
```

```
Algorithm mysteryRecursive($A[\ell\dots r]$)
  if $\ell == r$ then
    return $\ell$;
  else
    $i$ = mysteryRecursive($A[\ell\dots\lfloor\frac{\ell+r}{2}\rfloor]$);
    $j$ = mysteryRecursive($A[\lfloor\frac{\ell+r}{2}\rfloor+1\dots r]$);
    if $A[i] \le A[j]$ then
      return $i$;
    else
      return $j$;
    end if
  end if
```

a) [2 marks] What does the algorithm compute? Justify your answer.

b) [1 mark] What is the algorithmic paradigm the algorithm belongs to?

c) [1 marks] Write the recurrence relation (formula + base condition) for $C(n)$, which counts the number of array elements comparisons.

d) [3 marks] Solve the recurrence relation by the backward substitution method to obtain an explicit formula for $C(n)$ in n.

e) [1 mark] Write the complexity class that $C(n)$ belongs to using the Big-Θ notation.

Transaction t_i	Size s_i	Fee f_i
1	1	4
2	3	9
3	2	6
4	4	11
5	5	13

Table 1: A toy example of five transactions with their corresponding sizes and fees.

Problem 2 (8 marks, 1 page). **Profit maximisation in block mining - version 1**.

As the Bitcoin price has quadrupled in the past one year (9/2020-9/2021), you have made a decision of becoming a miner to earn some profit. You find out that in Bitcoin blockchain and the like, miners are responsible for constructing blocks and if successful (being the first to solve a puzzle), will receive not only a base reward but also transaction fees included in the transactions in the block. While the base reward is fixed and out of control of the miners, the transaction fees are not. You quickly figure out that one way for the miners to maximise their profit is to select the set of transactions that sum up to the highest fee. The problem formulation you come up with is: given n available transactions, in which transaction t_i has size s_i and pays fee f_i, $1 \leq i \leq n$, the miner should select a set I of transaction (indices) that has the maximum total fee $\sum_{i \in I} f_i$ while guaranteeing that the total size $\sum_{i \in I} s_i$ does not exceed the block size limit b.

a) [4 marks, 1/2 page] Design a greedy algorithm for this problem (note that it does NOT have to return the optimal solution): algorithm description (1 mark) + short pseudocode (1 mark) + complexity analysis (1 mark). Run it on the toy example in Table 1 with $b = 8$ and write down the list of transactions selected by the algorithm in their corresponding order (1 mark).

b) [4 marks, 1/2 page] Design a dynamic programming algorithm of complexity $O(nb)$ that can find a set of transactions that maximises the profit: write down the recursion formula (1 mark), build the dynamic programming table for the toy example in Table 1 with $b = 8$ (2 marks), and identify the set of transactions output by the algorithm together with the maximum profit (1 mark).

Problem 3. [10 marks, 3 pages] **The oracle and the mysterious answer**.

In the search for the meaning of life, you set off for a dangerous trip to visit a mysterious oracle living in a temple at the heart of the Great Victoria Desert. Barely escaping a terrible sand storm, you've finally found the oasis at a great cost: all the supplies have been buried under the sand and the only camel has run away. To make it worse, there is no sign of any source of water and all the plants surrounding the temple are dead. Apparently the oasis has been drying up for quite some time.

Staggering into the temple, surprisingly, you find the oracle, who seems to be waiting for you. "Speak, human! You can ask me one question." What a hoarse and emotionless voice! Is it because she has not had water for a long time? "Oracle, I... ", You hesitate. Before going, you had a list of all deep-meaning questions prepared. But now, what the heck! Licking your chapped lips, you ask "Oracle... where can I find water?" The oracle regards you for almost five seconds, and then asks "Computer scientist, yes?" "Y... yes," you reply, clearing your throat. She's an oracle after all. But you suddenly have this strange thought: what did the oracle do for a living before taking this... job? But surely you cannot ask a second question. The oracle immediately pulls out a piece of paper and a pen, writes something down and hands it to you eagerly as if she has been waiting to do this for a long time. Although it is rather dim in the temple, you can still feel her intense gaze under the veil. "The answer is in the solution," said the oracle, still with her emotionless voice. You turn toward the door to get some more light. It is a vaguely familiar problem about a compression algorithm. Not again!!! You groan and turn back to face the oracle, ready to shoot another question or at least, a complaint. But there is nobody there... But the unbearable thirst reminds you that you should not worry about the oracle's whereabouts. The answer for where to find the water is in the solution, and you need to crack the problem first. Is the oracle helpful? Or does she just want to play? There's only one way to find out...

Oracle's problem. There are ten letters whose frequencies are given in Table 2.

Letter	A	B	E	ϵ	H	N	O	T	U	Y
Frequency	27	9	12	10	1	11	5	7	16	2

Table 2: Frequencies of different letters for Problem 3.

a) [2 marks, 1 page] Construct a *min-heap* (tree) using the bottom-up heap construction to represent the frequencies in Table 2 (using the given order). Please use (label:frequency), for example, "H:1", for nodes in the heap. Describe (draw) all the steps required (use two-head arrows to indicate an exchange between two nodes). 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 the keys stored at its two children nodes.

b) [2 marks, 2/3 page] Illustrate (draw) the process of dequeuing the two letters of smallest frequencies from the heap *one by one* (dequeue -> repair -> dequeue -> repair).

c) [1 mark, 1/3 page] Illustrate (draw) the process of enqueuing the new element (i.e. enqueue -> repair), which results from the combination of frequencies of the two dequeued elements earlier. For example, if 'H:1' and 'Y:2' were dequeued, then the new element 'HY:3' would be enqueued.

d) [4 marks, 1 page] Provide a complete construction of the Huffman tree (3 marks) together with the codes for all letters (1 mark). Only the complete final tree needs to be given. However, each intermediate node must be labelled with both frequencies, e.g. "3", and the step in which it is constructed, e.g. "Step 1". When constructing the trees, the following rule applies: for every node from the root, the frequency of its left child must be smaller than or equal to that of its right child.

(Have you found out her answer? Does it help?)

2 Part II: Advanced

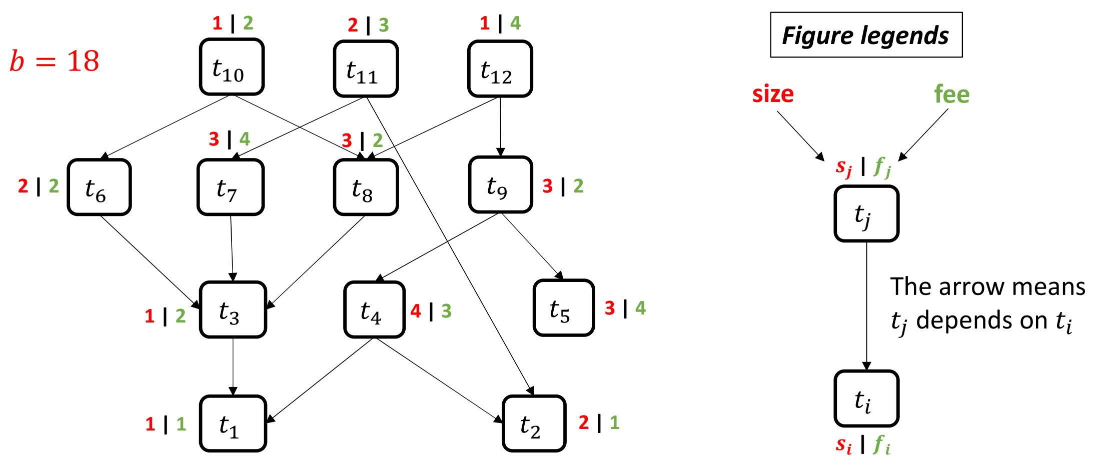


Figure 1: A toy example of twelve transactions with their corresponding sizes (red) and fees (green). For example, t_3 has size $s_3 = 1$ and fee $f_3 = 2$. The dependency among transactions is represented by arrows: $t_j \to t_i$ means that t_j depends on t_i. For instance, t_7 depends on t_3, which in turn depends on t_1. The block size limit is $b = 18$.

Problem 4 (8 marks, 1.5 pages)**. Profit maximisation in block mining - version 2***.
After executing your algorithms that select a set I of transactions from a pool of n unconfirmed transactions (waiting to be included in a block) that has total size $\sum_{i \in I} s_i \le b$, where b is the *block size limit*, while achieving maximum total fee $\sum_{i \in I} f_i$, you discovered a problem with this formulation. It dawned on you that the transactions are NOT independent of each other. For instance, if Alice pays 1 bitcoin to Bob in Transaction t_i, who in turn uses this coin to pay Carl in Transaction t_j, then t_j will depend on t_i. More specifically, in Bitcoin, t_j depends on t_i if an output of t_i is used as an input of t_j (Figure 2). Hence, to include t_j in the current block, t_i must be either already included in an existing block or in the same block as t_j. This is the Dependency Rule. Note that t_i in its turn may depend on other transactions, which also need to be included, and so on.

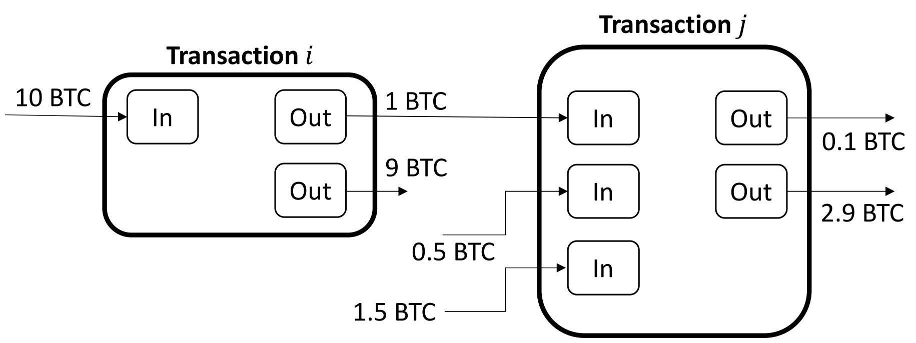


Figure 2: Dependency among Bitcoin transactions. Transaction j uses one output of Transaction i as its input. In such a case, Transaction j depends on Transaction i.

Suppose that you have run a preprocessing algorithm that returns the *dependency list* L_j that comprises of all i where t_j depends on t_i, for each $1 \le j \le n$. For instance, in the toy example in Figure 1, $L_{10} = \{6, 8\}$ and $L_{11} = \{2, 7\}$.

a) [5 marks, 1 page] Design an efficient algorithm (worst-case complexity $O(n^2)$) that takes as input the number of transactions n, the sizes s_i and fees f_i, the dependency lists L_i, $i = 1,\dots,n$, and an index set $J \subseteq \{1,2,\dots,n\}$, and returns the list T_J ($J \subseteq T_J$) of ALL the transactions (indices) that must be included if the transactions $\{t_j : j \in J\}$ are to be included in a block, so that the Dependency Rule is respected. For example, when $J = \{4,7\}$, we have $T_J = \{1,2,3,4,7\}$. Note that T_J must contain J as a subset. The solution must include:

- (2 marks) algorithm description,
- (1 mark) short pseudocode, and
- (1 mark) complexity analysis, and
- (1 mark) the list of transactions T_J output by the algorithm applied to the toy example in Figure 1 with $J = \{5,10,11\}$ and the corresponding total size & fee.

b) [3 marks, 1/2 page] Based on Part a), design an exhaustive search (algorithm) that returns a set of transactions (indices) J^* to form a block that respects the Dependency Rule and has total size at most b while maximising the total transaction fee. Input to the algorithm: n, b, s_i, f_i, L_i, $i = 1,\dots,n$. The solution must include:

- (1 mark) algorithm description, and
- (2 marks) the set of transactions (indices) J^* output by the exhaustive search, its total size and its (maximum) total profit for the toy example in Figure 1.

Problem 5 (6 marks + 1 bonus mark, 2 pages). **Perfect Binary Tree Partition****.

A *perfect binary tree* is a rooted binary tree in which every non-leaf node has exactly two children and all the leaves are at the same depth. Note that a binary tree of height h will have exactly 2^h leaves. Given a perfect binary tree with height h, let $\mathscr{S}_h = \{2, 3, \ldots, 2^{h+1} - 1\}$ denote the set of all $2^{h+1} - 2$ nodes of the tree excluding the root. Let $S_1, \ldots, S_h$ be a partition of $\mathscr{S}_h$, that is, $S_i \cap S_j = \varnothing$ for all $i \neq j$, and $\cup_{1 \le i \le h} S_i = \mathscr{S}_h$. A partition is called *unrelated* if each set S_i in the partition doesn't contain two nodes in which one is an ancestor of the other (if $u \neq v$ and u lies on the (unique) path from the root to v then u is called an ancestor of v while v is called a descendant of u). We define the balance index of a partition $(S_1, \ldots, S_h)$ as the difference between the maximum size and the minimum size of a set in the partition. More formally,

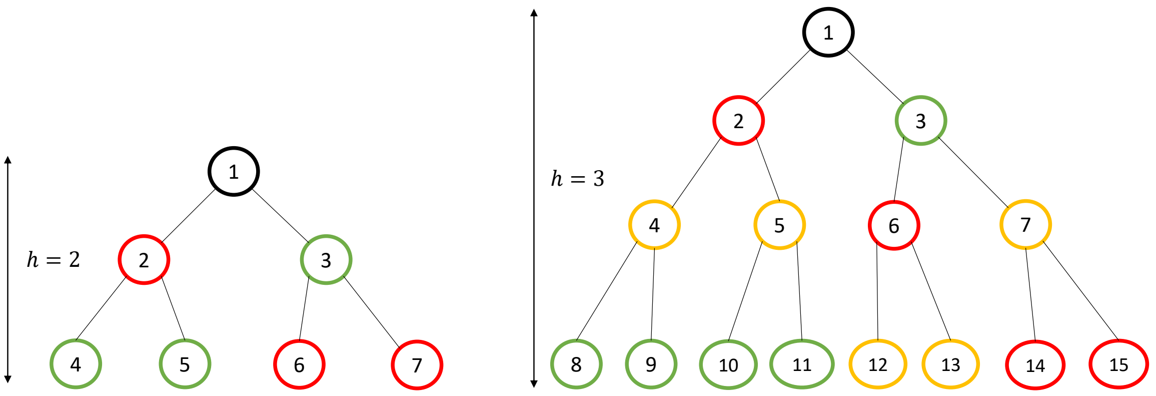
$$b(S_1, \ldots, S_h) \triangleq \max_{1 \le i \le h} |S_i| - \min_{1 \le i \le h} |S_i|.$$

a) [5 marks, 1 page] Design an efficient algorithm (algorithm description, pseudo code, and an estimated time complexity if possible) that finds an unrelated partition of $\mathscr{S}_h$ that has minimum balance index. The algorithm is deemed efficient if the submitted implementation (Java/Python) can find an unrelated partition with balance index 0 or 1 for each $h \le 10$ in less than five seconds. *Hint: iterative improvement*.

b) [1 marks] Applicable if the algorithm can find an unrelated partitions with balance index 1 for $h = 14$ in less than 10 hours.

c) [1 bonus mark, 1 page] Provide a mathematical proof that the algorithm developed in Part a) can find an unrelated partition with balance index 0 or 1 for ALL $h \ge 2$.

See the examples in Figure 3 for unrelated partitions of balance index 0/1 when $h = 2, 3$. There can be other partitions satisfying the requirement. More details on how to prepare the solution for Problem 5 will be updated on Assignment 2 Queries (Discussion Forum).



S_1 (RED)	2, 6, 7
S_2 (GREEN)	3, 4, 5

S_1 (RED)	2, 6, 14, 15
S_2 (GREEN)	3, 8, 9, 10, 11
S_3 (YELLOW)	4, 5, 7, 12, 13

Figure 3: Unrelated partitions with balance index 0 for $h = 2$ and balance index 1 for $h = 3$. Each set S_i in a partition consists of nodes with the same color.

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 discussion forum on Canvas, 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.