# The University of Melbourne

# COMP20007: Design of Algorithms, Semester 1 2022 - Exam

| | |
|---|---|
| **Reading Time** | 15 minutes. |
| **Writing Time** | Three hours. |
| **Submission Time** | 30 minutes. |

## Instructions to Students

- This exam counts for 60% of your final grade.
- You have a total of 3:15h to finish the exam. 15min as reading time and 3h to answer the exam questions. **Read the exam before you start answering questions.**
- You should allow at least 30 minutes to scan and upload your solutions. This is in addition to the 3:15h allocated for the exam.
- **Important: if you upload your submissions on Gradescope during this additional 30min window, Gradescope will flag your submission as LATE. This can be ignored: as long as your submission does not go over this 30min window your submission will not incur a late penalty. Any emails asking about this will be ignored.**
- Submissions after this window will incur in a late penalty (-1 mark per minute up to 30 minutes, after which late submissions are closed) and will need to be made using the following link:

  `https://unimelbcloud-my.sharepoint.com/:f:/g/personal/lkulik_unimelb_edu_au/EgsjRLThFDBOoG-sjMa-0L0Boyq2NQo1YDiaEyQjzIE1mQ`
- The test is open book, which means you may use course materials provided via the LMS or the text book **but must not use any other resource including the Internet.**
- **You must not communicate with other students or make use of the Internet.**
- Solutions must be written on separate pieces of paper with pen or pencil. You must write your solution to each question on a new sheet of paper and clearly indicate which question you are answering on each page.
- You must not use a tablet to complete this test.
- You must indicate which question is answered on which page via Gradescope.
- You must use a Scanner app on your smartphone to scan your solutions, email them to your laptop and then submit a PDF file to Gradescope via Canvas.
- A Gradescope guide to scanning your test can be found here:

  `https://gradescope-static-assets.s3-us-west-2.amazonaws.com/help/submitting_hw_guide.pdf`
- Exam support is available via `https://canvas.lms.unimelb.edu.au/courses/124591/conferences`

## Common Facts and Notation

We know from the lectures that for $0 < \varepsilon < 1 < c$: $1 \prec \log n \prec n^{\varepsilon} \prec n^{c} \prec n^{\log n} \prec c^{n} \prec n^{n}$.

Remember that $\log n$ is an abbreviation for $\log_2 n$, the binary logarithm.

Remember also that for $a > 0$: $\sum_{i=0}^{n} (a^i) = \dfrac{1 - a^{n+1}}{1-a}$.

We remind you also of the following conventions:

$$\sum_{i=m}^{n} (s_i) = s_m + s_{m+1} + \ldots + s_{n-1} + s_n \quad \text{and} \quad \prod_{i=m}^{n} (p_i) = p_m \cdot p_{m+1} \cdot \ldots \cdot p_{n-1} \cdot p_n.$$

Logarithm laws:

$$\log a + \log b = \log ab, \quad \log a - \log b = \log a/b \quad \text{and} \quad \log a^n = n \log a$$

.

$$a^{\log_b c} = c^{log_b a} (a > 0, b > 0, c > 0)$$

.

## Master Theorem

For integer constants $a \geq 1$ and $b > 1$, and function $f$ with $f(n) \in \Theta(n^d), d \geq 0$, the recurrence

$$T(n) = aT(n/b) + f(n)$$

(with $T(1) = c$) has solutions, and

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Note that we also allow $a$ to be greater than $b$.

## Question 1 [5 marks]

**(a)** Given an integer array $A$, with $n$ elements, what is the complexity of the following function?

```
function UNKNOWN($A, n$)
    for $j \leftarrow 0$ to $n - 2$ do
        if $A[j] > A[j+1]$ then
            SWAP($A[j], A[j+1]$)
```

**(b)** Given an integer array $A$, with $n$ elements, what is the complexity of the following function? Notice it calls the UNKNOWN function defined in Part **(a)**.

```
function UNKNOWN2($A, n$)
    for $i \leftarrow 0$ to $n - 2$ do
        UNKNOWN($A, n - i$)
```

**(c)** Rank the following functions by increasing order of growth from lowest to highest:

$$n^3, (\log_3 n)^3, \log_3(n^3), 27^{\log_3 n}, (\log_3 n)^{\log_3 n}, n^{0.003}.$$

Write your answer in a table format, using one line for each function unless they have the same order of growth. In this case, write them in the same line. For example if the functions were $n$, $n^2$, $2n$, then you answer would be:

| |
|---|
| $n$, $2n$ |
| $n^2$ |

**Hint:** start by finding the function with lowest order.

## Question 2 [8 marks]

Professor Fast has to check a large number of arrays. The arrays are supposed to contain temperature readings but sometimes the sensor does not work and instead reports a blank space "_" instead of a numeric value. Given an array, Professor Fast wants to find if all elements are valid readings (no blank spaces). Since there is a large number of arrays, Professor Fast decides that a simple linear scan is not efficient.

The algorithm for checking one array works as follows: to search for a blank space in an array $A$ of unsorted elements, we compute the index *mid* of the middle element as we have done in the lectures for binary search. If the middle element $A[mid]$ is a blank space, we stop the search and return "False". Otherwise, we split the array $A$ at the middle element into two subarrays $A_l$ and $A_r$, where the last index of $A_l$ is $mid-1$ and the first index of $A_r$ is $mid+1$. Note $|len(A_l) - len(A_r)| \leq 1$, that is, the subarrays have roughly the same size. We recursively search for the blank space in the subarrays $A_l$ and $A_r$. If no blank space is found return "True", otherwise return "False". Here is the complete pseudocode:

```
function CLEAN($A[], lo, hi$)
    if $lo > hi$ then
        return $True$
    $mid \leftarrow lo + (hi - lo)/2$
    if $A[mid] =$ "_" then
        return $False$
    else
        return CLEAN($A, lo, mid-1$) and CLEAN($A, mid+1, hi$)
```

**(a)** Clearly state your basic operation and derive the recurrence relation for this algorithm, including the base case.

**(b)** Derive the complexity of this algorithm using the Master Theorem.

**(c)** Derive the complexity of this algorithm through telescoping.

## Question 3 [8 marks]

**(a)** Can you use Depth-First Search to compute the shortest path in an arbitrary tree? Explain your answer.

**(b)** Can you use Depth-First Search to compute the shortest path in an unweighted graph? Explain your answer.

**(c)** Assume $G$ is a connected, undirected, unweighted graph. The eccentricity $ecc(V)$ of a vertex $V$ is the maximum distance from $V$ to any other vertex in $G$. Outline an efficient algorithm to compute $ecc(V)$ for a given vertex $V$. For full marks, your algorithm must be linear in the number of vertices.

## Question 4 [6 marks]

In this task you will write a function in pseudocode to compute the *perfect split* of an array. The perfect split index $i$ of an array is the index which causes the sum of elements with an index less than $i$ to be equal to the sum of elements with an index greater than $i$. The element at index position $i$ is not included. More formally: $A[0] + A[1] + \ldots + A[i-1] = A[i+1] + \ldots + A[n-1], 0 < i < n-1$. The array $A$ consists of $n$ integer values, which are not necessarily distinct.

**Example:**

Input: $A[] = [-6, 1, 4, 2, -3, 1, 1]$

Output: 3

Reason: $A[0] + A[1] + A[2] = -1 = A[4] + A[5] + A[6]$

Write pseudocode for an algorithm that takes the array $A$ as its argument and returns the perfect split index $i$ if it exists, $-1$ otherwise. Discuss the time and space efficiency of your algorithm. For full marks, your algorithm needs to run faster than $\Theta(n^2)$ time.

## Question 5 **[8 marks]**

This question covers string matching and compression.

**(a)** Suppose you have a string $S$ containing 104 characters, composed of 100 letters $a$ in sequence and followed by the sequence *abba*. Your goal is to find the pattern *abba* in this string. How many character comparisons would be required if you use Horspool's algorithm to find the pattern in this case? Show your calculations.

**(b)** Suppose you have an alphabet $A$ with $m$ symbols, where $3 \leq m \leq 26$ and each symbol corresponds to the first $m$ lowercase letters of the English alphabet. For example, if $m = 3$, then $A = \{a, b, c\}$. Describe the best *fixed-size encoding* (in terms of space required to store each symbol) for an alphabet $A$.

**(c)** Assume again we have an alphabet $A$ as defined in Part **(b)**, where $m = 8$. Given a string $S'$, let $f_i$ be the frequency of symbol $a_i$ in the string $S'$ (so, $f_1$ is the frequency of $a$, $f_2$ is the frequency of $b$, etc). Suppose the frequencies of each character in $S'$ follow this rule:

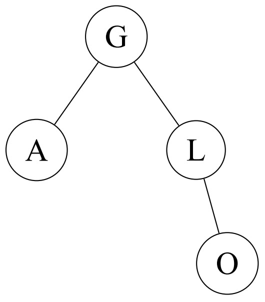
$$f_i > \sum_{j=i+1}^{m} f_j \quad \text{for all} \quad 1 \leq i < m$$

Draw a Huffman code tree that gives the best encoding for the string $S'$. Briefly explain your answer.

## Question 6 [9 marks]

This question covers Binary Search Trees and sorting.

**(a)** Suppose you have a dictionary implemented via an AVL-Tree. Here is the tree after inserting the letters `A, L, G, O`:



Insert the letter `R` into the tree above, draw the resulting tree and write down the balance factor for each node. Then, peform any necessary rotations to rebalance it and draw the resulting, balanced AVL-Tree. You should show each step necessary for rebalancing.

**(b)** Suppose now your dictionary is implemented via a 2-3 Tree. Insert the following letters in your 2-3 Tree, in this order (from left to right):

`D, E, S, I, G, N`

Draw the tree after each insertion step, including any required operations to keep the tree valid.

**(c)** Perform Counting Sort on the following array. Show your work, including the count array.

$$\left[ 3, 1, 8, 1, 3, 9, 8, 1, 6 \right]$$

**(d)** The University of Melbourne stores a list of students for every subject. Each student has a *name* and a *surname*. The subject coordinator can see the list of students in a web application. They would like to see the full list in alphabetical order, starting by name and then by surname. Here is an example, containing 7 students:

**Unsorted**

| *Name* | *Surname* |
|---|---|
| Ashley | Zen |
| Billie | Yang |
| Cam | Xu |
| Cam | Zen |
| Ashley | Xu |
| Billie | Xu |
| Ashley | Yang |

**Sorted**

| *Name* | *Surname* |
|---|---|
| Ashley | Xu |
| Ashley | Yang |
| Ashley | Zen |
| Billie | Xu |
| Billie | Yang |
| Cam | Xu |
| Cam | Zen |

As the web developer responsible for this application, you came with the following algorithm to sort this list, where *S* corresponds to the list of students as in the example above:

```
function SortStudents($S$)
    $S \leftarrow$ Sort($S$, surname)
    $S \leftarrow$ Sort($S$, name)
    return $S$
```

In this pseudocode, Sort is a function that sorts the list of students by a specific *key* and returns the sorted list. In the first call, it sorts all students by surname only, while in the second call it sorts the students by name only.

Your task is to choose an appropriate algorithm to implement Sort. Given it is a light web application, the algorithm cannot use extra memory (recursion is allowed). Which algorithm would you choose, so the overall SortStudents procedure works as intended? Justify your answer.

## Question 7 **[8 marks]**

Daniel has baked a lot of muffins and wants to pack them into boxes. He has access to a collection of boxes of $K$ different sizes $s_1 < s_2 < \ldots < s_K$. A box of size $s_k$ must hold exactly $s_k$ muffins. Daniel wants to use the minimum number of boxes to pack $M$ muffins. Assume that he has access to an unlimited number of boxes of each size, and that $s_1 = 1$.

For example, assume your collection of boxes is $\{s_1 = 1, s_2 = 3, s_3 = 7\}$. In this case, the minimum number of boxes to pack $M = 5$ muffins is 3: two $s_1$ boxes and one $s_2$ box. For $M = 6$, it is 2: two $s_2$ boxes.

**(a)** Suppose that Daniel uses a greedy approach, where he iteratively chooses the largest possible box. That is, Daniel finds the largest $k$ such that $s_k$ is not greater than the number of currently unpacked muffins, packs $s_k$ muffins into a box, and repeats the procedure. Explain with an example why this greedy approach does not produce the optimal solution in general. You can choose the values of $K, M, s_2, \ldots, s_K$ to give an example.

**(b)** Let $g(M)$ be the minimum number of boxes needed to pack $M$ muffins. Write down a recurrence relation for $g(M)$, including the appropriate base case.

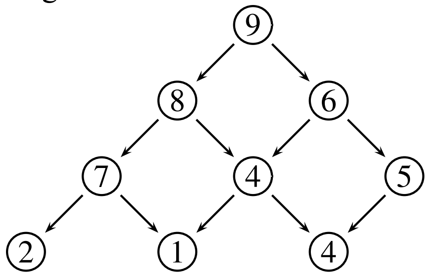
**(c)** What is the time complexity of a bottom-up dynamic programming algorithm to find $g(M)$? Justify your answer. You can write a pseudocode solution to help justify your answer but that is not required.

## Question 8 [8 marks]

A *web* is a data structure similar to a heap that can be used to implement a priority queue. As with a heap, a web is defined by two properties:

- **Structure property:** A web $W$ consists of $l$ levels, $l > 0$. The $i$th level, for $0 \leq i < l$, contains at most $i+1$ nodes, indicated as $W_{i,j}$ for $0 \leq j \leq i$. All levels but the last are completely filled, and the last level is left-justified.
- **Ordering property:** Any node $W_{i,j}$ has at most two *children*: $W_{i+1,j}$ and $W_{i+1,j+1}$, if those nodes exists. The priority of a node is always greater than or equal to the priority of either child node.

For example, the following diagram shows a web with 9 nodes and 4 levels.



A web $W$ with $n$ nodes can be stored in an array $A$ of size $n$, similarly to an array-based heap. So, for example, if $n \geq 1$, the root node of the web $W_{0,0}$ will be stored at $A[0]$. This means we can access nodes in a web using a mathematical formula, in the same we use for heaps. Assume $n$, $i$ and $j$ are such that all indicated web nodes actually exist.

**(a)** For a web $W$ with $l$ levels ($l \geq 1$), give the upper and lower bounds for the number of nodes $n$. For example, a web with 3 levels has at least 4 and at most 6 nodes. Show your work in detail.

**(b)** Give the formula to access node $W_{i,j}$. Justify your answer. You can use the result from **(a)** to help justifying.

**(c)** Briefly describe how to eject the maximal element (the root node) from a web $W$. Find the complexity of your approach and justify your conclusion.

**END OF EXAM**