School of Computing and Information Systems
COMP20007 Design of Algorithms
Semester 1, 2020
End-of-Semester Test

Instructions to Students

- The total time to read, complete, scan and upload your solutions to this test is 1 hour. You should allow at least 15 minutes to scan and upload your solutions.
- You must submit your solutions by 10:00 AM AEST.
- Late submissions will be subject to a 10% penalty per 5 minutes late and submissions after 10:20 AM AEST will not be marked.
- This test contains 4 questions which will be marked out of 20 marks.
- This test will count 20% towards your final mark if your score in this test is higher than your score in the Final Exam.
- The test is open book, which means you may only 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 also 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`
- The Dropbox and Microsoft OneDrive mobile apps also have scanning capabilities.

Question 1 [5 Marks]

We know, from lectures, the following facts. For $0 < \epsilon < 1 < c$,

$$1 \prec \log n \prec n^{\epsilon} \prec n^{c} \prec n^{\log n} \prec c^{n} \prec n^{n},$$

where $f(n) \prec g(n)$ means both $f(n) \in O(g(n))$ and $g(n) \notin O(f(n))$ (i.e. $g(n)$ is not in $O(f(n))$).

For the following pairs of functions, $f(n), g(n)$, determine if $f(n) \in \Theta(g(n)), f(n) \in O(g(n))$, or $f(n) \in \Omega(g(n))$, making the strongest statement possible. That is, if both $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$ are true, you **must** answer with the form $f(n) \in \Theta(g(n))$. You must answer with $f(n)$ on the left and $g(n)$ on the right, for example, you may not answer $g(n) \in O(f(n))$.

You **must** show **all** working and provide sufficient **justification**. A correct answer that does not show your working will result in 0 marks.

(a) $f(n) = \sqrt{16n^{16}}, \quad g(n) = 4n^5 + 10n$

(b) $f(n) = \log_2(3n), \quad g(n) = \log_2(n^3)$

(c) $f(n) = 3^{2n}, \quad g(n) = 3^{n+1}$

For the following question you must **solve the recurrence relation**, provide a **closed form expression** for $T(n)$ and say which **Big-Theta** class $T(n)$ belongs to.

(d) $T(n) = T(n-1) + 4n, \quad T(0) = 20007$

Question 2 [6 Marks]

For **all** of the following questions, you **must** explain your reasoning. Your answer must demonstrate that you understand your solution and have considered all cases.

For the purpose of this question assume all elements in a min-heap are distinct.

(a) Explain the restrictions on where the second smallest element in a min-heap can be found.

You **must** justify your response, explaining **why** the element can be found there.

(b) What is the time complexity of locating (but not removing) the third smallest element in a min-heap?

As part of your response you **must** describe and justify where the third smallest element can be found. Give the tightest possible bound using O, Ω, or Θ, and justify your response.

(c) A heap can be stored in array with where a node at index i has its left and right children in indices $2i$ and $2i + 1$ respectively.

Is this possible for AVL trees? Explain why or why not.

Question 3 [4 Marks]

Use Huffman's algorithm to construct a code tree based on the letter frequencies in the string:

`assessmentteam`

When constructing the tree you should sort the letters in ascending frequency order, breaking ties alphabetically.

You must show all working and the code tree you have constructed.

(a) List the codes for each of the letters `"a"`, `"e"`, `"m"`, `"n"`, `"s"`, and `"t"`.

You should assume that the left branches in the code tree should be `0`s, and the right branches should be `1`s.

(b) Use the codes you have generated to give an encoding for:

`state`

What is the length, in bits, of your encoded string?

Question 4 [5 marks]

AssumptionSearch is an algorithm which searches for a numerical key k in an array A of length n which is **sorted in ascending order** by making the assumption that the contents of the array are spread out relatively evenly.

Like binary search, AssumptionSearch compares k with an element of the array $A[m]$ and—depending on whether or not $k < A[m]$ or $k > A[m]$—recursively searches for k in the left or right components of the array.

Rather than selecting m in the middle of the array, however, AssumptionSearch looks at the highest and lowest values (*i.e.,* $\ell = A[0]$ and $h = A[n-1]$) and says "if k is $X\%$ of the way between ℓ and h then we will choose the index m to be $X\%$ of the way between 0 and $n-1$". You should use integer division for index computations, as in binary search.

(a) Illustrate the execution of AssumptionSearch on the following input:

$$A = [0, 5, 10, 11, 12, 18, 20, 30, 40] \qquad k = 30$$

How many different values of $A[m]$ are compared with k?

(b) Write the pseudocode for AssumptionSearch using the following function signature:

func AssumptionSearch$(k, A[0 \ldots (n-1)])$

Your algorithm must return the index of the matching element or return NotFound.

(c) Using Big-Theta notation describe the worst-case time complexity of AssumptionSearch and give a **concrete example** (*i.e.,* give an array A and a key k) of an input that will result in this worst-case behaviour.

END OF TEST