School of Computing and Information Systems
COMP20007 Design of Algorithms
Semester 1, 2021
Mid Semester Test - Solutions

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.
- This test contains 4 questions which will be marked out of 10 marks.
- This is an open book test. You may consult resources made available via the LMS or the text book.
- You must not communicate with other students or make use of the Internet.
- Solutions must be written on separate pieces of paper.
- 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 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 [2 Marks]

We know the following facts about complexity bounds. 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. A correct answer that does not show your working will result in 0 marks.

(a)	$f(n) = n^3$,	$g(n) = (n+2)^3$
(b)	$f(n) = 3^{(2n)}$,	$g(n) = 3^{(n+2)}$
(c)	$f(n) = (3n)^2$,	$g(n) = 3^{n^2}$
(d)	$f(n) = (\log_2 n)^{\log_2 n}$,	$g(n) = \sqrt{n}$

*(d) is more difficult than the other three parts (a) to (c). Make sure you answer the other questions first if you get stuck on (d)!

Solution

(d) $(\log_2 n)^{\log_2 n} > 2^{\log_2 n} = n$. So $f(n) \in \Omega(g(n))$.

Question 2 [2 Marks]

The following algorithm can be used to compute the distance between the two closest elements in an array of numbers, that is, to compute $min_{i \neq j}|a_i - a_j|$:

```
function MinDistance(a_{0..n-1})
    dmin ← ∞
    for i ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
            if |a_i − a_j| < dmin then
                dmin ← |a_i − a_j|
    return dmin
```

(a) Give the strongest possible claim about the runtime complexity of the above algorithm.

(b) Briefly describe a better (in term of runtime complexity) algorithm for the above task.

Solution

(a) The main operation is testing the condition inside the **if** statement. That operation runs for $\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = n(n-1)/2$ times. It's $\Theta(n^2)$.

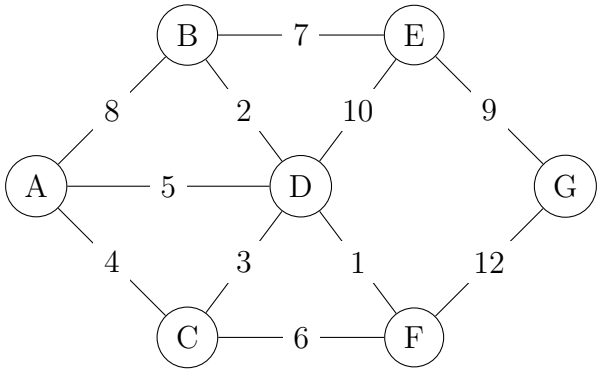
(b) One possible solution is:

1. Sort a in increasing order using mergesort
2. Return $\min(a_{i+1} - a_i)$ with $i = 1..n-2$

Since the second step is linear to n, and the mergesort algorithm employed in the first step has the complexity of $\Theta(n \log n)$, the above algorithm has the complexity of $\Theta(n \log n)$. That is better than $\Theta(n^2)$ of the first algorithm because $\log n \prec n$.

Question 3 [4 Marks]

A city council is building a network of dedicated bicycle tracks that connects all of its major facilities (such as council offices and libraries). A preliminary study for the cost of possible tracks between the facilities is summarized in the graph below, where the nodes represent the facilities, and the edges represent the possible tracks accompanied by the building costs. Note that there is only a relative small number of possible tracks. The other tracks are just impossible to build or would have extremely high cost.



The council wants to build a network connecting all facilities.

(a) Starting from node A, perform a Depth-First Search in the graph. When choosing between multiple nodes, you have to choose the node corresponding to the **edge with lowest cost** (as long as the node is unmarked). Write down the order you traverse the nodes.

(b) Starting from node A, perform a Breadth-First Search in the graph. When choosing between multiple nodes, you have to choose the node corresponding to the **edge with lowest cost** (as long as the node is unmarked). Write down the order you traverse the nodes.

(c) The council wants to build the network with minimal cost. Could you use one of your solutions above using DFS or BFS to solve this problem? Justify your answer.

(d) Suppose that the network has already been built, and the council now has 2 newly built facilities X and Y. The council is supplied with a list of the projected costs for the bicycle track between X and Y (if the track is possible), as well as for all the possible tracks between X and Y to the old facilities. There is at least one possible track from X, and at least one from Y, to the old facilities. Describe an algorithm to extend the current bicycle track network to cover the two new facilities with minimal additional cost. What is the complexity of this algorithm?

Solution

(a) A C D F G E B

(b) A C D B F E G

(c) No, because the traversal via DFS or BFS does not guarantee the total edge cost will be minimal. The desired network should be obtained via a Minimum Spanning Tree algorithm.

(d) We need only 2 more tracks to connect both X and Y to the network, and we need to choose the two tracks that have minimal total costs. Algorithm:

1. Choose a lowest cost track tX from the possible tracks from X to the old facilities
2. Choose a lowest cost track tY from the possible tracks from X to the old facilities
3. Amongst tX, tY and track XY (if existed), choose the two tracks with smallest costs and add them to the existing network.

Complexity is $O(n)$ where n is the number of facilities.

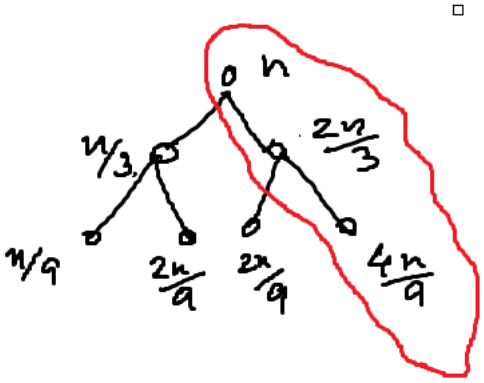
Question 4 [2 marks]

(a) Binary search divides an array into two (roughly) equal halves in each step when searching for a key. If we divide the array into two arrays in each step, one which is roughly 1/3 of its size and another one that is roughly 2/3 of its size, what will be the worst-case complexity for this modified search algorithm? Justify your answer.

(b) Use a recurrence relation to determine the complexity of the function RECUR below. Assume the addition operation (+) is $\Theta(1)$.

```
function RECUR(n)
    if n ≤ 1 then
        return 1
    else
        return n+RECUR(n/5)
```

Solution



(a)

While searching for a match, the right one is the longest branch, so we need to consider its height in worst case. So, $(2/3)^k n = 1$, $k = log_{3/2} n$.

(b)

$$
\begin{aligned}
T(1) &= b \\
T(n) &= T(n/5) + a \\
T(n/5) &= T(n/5^2) + a \\
T(n/5^2) &= T(n/5^3) + a \\
T(n) &= T(n/5^2) + 2a \\
&= T(n/5^3) + 3a \\
&\dots \\
T(n) &= T(n/5^k) + ka
\end{aligned}
$$

Setting $k = log_5(n), T(n) = T(1) + alog_5(n) = b + alog_5(n)$ Thus, the time complexity is $O(log(n))$.