NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
SEMESTER II (2018-2019)

MOCK FINAL EXAM FOR
CS2040: DATA STRUCTURES AND ALGORITHMS
SUGGESTED ANSWERS

24 April 2019, 1pm
Time Allowed: 2h

MATRICULATION NUMBER:

A	0	1						

INSTRUCTIONS TO CANDIDATES
1. Write your matriculation number in the space provided above.
2. This examination paper consists of **10 MCQ questions and 2 structured questions** and comprises **ten (10)** printed pages including this front page.
3. Answer the MCQ questions by shading the correct option on the OCR form and the programming questions directly in the space given after each question. If necessary, ask for an extra sheet of paper and attach it at the end of the test
4. Marks allocated to each question are indicated. Total marks for the paper is 100.

EXAMINER'S USE ONLY			
Questions	**Possible**	**Marks**	**Check**
1 - 10	30		
11	15		
12	55		
Total	100		

Prepared by Wang Zhi Jian. Some questions taken from materials used in CS2040 AY18/19 Sem 1.

Section 1. MCQ Questions (3 marks each)

1. Given a hash table of fixed size *m*, load factor *α*, hash function hash(*key*) = *key* mod *m* and collisions resolved using quadratic probing, for which values of *m* and *α* below will quadratic probing definitely terminate?

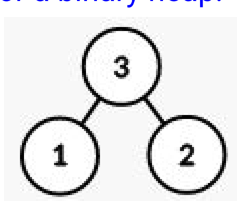
    A. *m* = 15, *α* = 0.3
    B. *m* = 15, *α* = 0.5
    C. *m* = 17, *α* = 0.3
    D. *m* = 17, *α* = 0.5
    E. *m* = 17, *α* = 0.7

    Refer to Lecture 8 slides.

2. Which of the following statements are **always true** for a binary max-heap?

    I. The largest element is positioned at the rightmost leaf node.
    II. For every node with two children, the value of the left child must be greater than that of the right child.
    III. When a binary max-heap is implemented using an array, the array is sorted from largest to smallest.
    IV. Given a heap of *n* elements, you can output a sorted list of values in O(*n*) time.

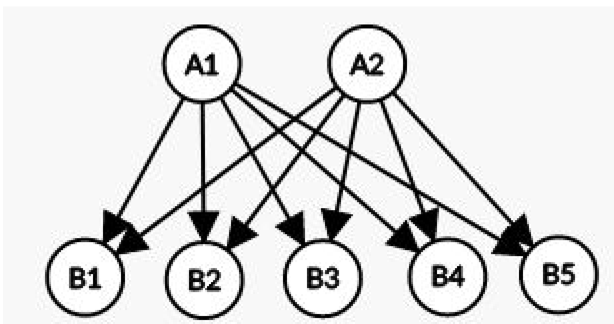
    A. IV only
    B. I and III only
    C. II and IV only
    D. III and IV only
    E. None of the above

    I. False: Largest element is positioned at the root node.
    II. False: No requirement for the relative ordering of the left child and right child of any node for a binary heap.
    III. False: 

is stored in an array as 3 1 2.
    IV. False: *n* calls to extract-max are needed, giving a running time of O(*n* log *n*).

3. Given a directed acyclic graph with n nodes where it is possible for exactly 2 nodes to reach all the other $n - 2$ nodes in the graph, what is the maximum possible number of topological orderings of the graph?

   A. $n - 2$
   B. n
   C. $(n - 2)!$
   D. $n!$
   E. None of the above



There are at least $2! \times (n - 2)!$ valid topological orderings.

4. Given the following hash function hash(*key*) = (3 × *key*) mod 13, which of the following is the resulting hash table (of size 13) after the following sequence of operations if linear probing is used? The leftmost slot has index 0.

add 5, add 1, add 18, add 14, add 3, add 10, del 14, add 12

A.

		5	1	18	10	DEL			3	12		

B.

		5	1	18	DEL	10				3	12	

C.

		5	1	18	14	10			3	12		

D.

		5	**1**	**18**	**DEL**	**10**			**3**	**12**		

E.

		5	1	10	DEL	18			3	12		

Refer to the Lecture 8 slides, trace/simulate carefully.

5. What's the tightest Big-*O* time complexity for the following segment of code?

```
public void BigO(int n) {
     if (n == 0) return;
     int k = n;
     for (int i = 0; i < n; i++) {
          k += n;
          for (int j = 0; j < n; j++) {
               k--;
          }
     }
     BigO(k/2);
}
```

A. O(log n)
B. O(n)
C. O(n log n)
D. O(n^2)
E. O(n^2 log n)

$n^2 + n^2 / 4 + n^2 / 16 + \ldots \le 2n^2 = O(n^2)$. Refer to Tutorial 5 Appendix slides.

6. Which of the following statements is **true**?

   A. Given an AVL tree, you can output a sorted list of all the keys in O(n) time using pre-order traversal.
   B. An AVL tree can be converted into a max heap in O(n) time.
   C. The balance factor of any node in a balanced AVL tree must be between 0 and 1.
   D. At most log n rotations are required after inserting a node into an AVL tree.
   E. The predecessor of any node in an AVL tree, if exists, can be found in O(1) time.

   A. False: In-order traversal is required.
   B. True: First perform any traversal on the tree to extract all the values in the AVL tree to an array. Then, heapify the array.
   C. False: Between -1 and 1
   D. False: At most a constant number of rotations is required.
   E. False: O(log n) time is required.

7. Given a graph with V vertices and E edges stored in an adjacency matrix, what is the tightest time complexity of DFS, Bellman Ford and Floyd Warshall respectively?

   A. O(V+E), O(VE), O(V^3)
   B. O(V+E), O(E^2), O(V^3)
   C. O(V^2), O(VE), O(V^3)
   D. O(V^2), O(E^2), O(V^3)
   E. None of the above

   Slight ambiguity in this question - you should run the algorithms on the adjacency matrix without first converting the graph to an adjacency list. Apologies!

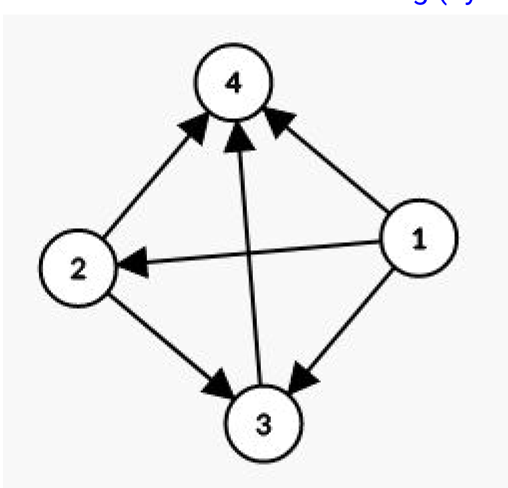
   DFS takes O(V^2) time, Bellman Ford takes O(V^3) time and Floyd Warshall takes O(V^3) time.

8. Which of the following statements is **true**?

A. In a connected graph with *N* vertices and *N* edges, relaxing all the edges in only one iteration of Bellman Ford is sufficient to detect a negative cycle.
B. In a directed, connected graph with *N* vertices and *N* edges, a topological ordering of the vertices definitely exists.
C. There exists a directed complete graph with more than 3 vertices such that a valid topological ordering exists.
D. If there are negatively weighted edges in a graph, then it is impossible for any variant of Dijkstra's Algorithm to solve the Single Source Shortest Path problem.
E. None of the above.

Some ambiguity in this question too - Option C wasn't phrased in a way that conveys what it
actually means. Apologies once again!

A. False: At least 2 iterations needed.
B. False: Direct the edges in a way such that a cycle exists in the graph.
C. True: The original phrasing of this option was "There exists a valid assignment of directions to the edges in a directed graph with at least 3 vertices such that a valid topological ordering exists." That was long, so I tried to shorten it, but the shortened
version had a different meaning (by accident). Anyway, a valid graph is as follows:



D. False: Modified Dijkstra's Algorithm works, for example.

9. To heapify an array of 8 elements, what is the maximum number of comparisons needed?

A. 8
B. 9
C. 10
D. 11
E. 12

10. Which of the following statements are **true**?

I. Quicksort is an in-place sorting algorithm
II. Selection Sort is a stable sorting algorithm
III. Insertion Sort can be simulated using one queue.
IV. Selection Sort can be simulated using one queue.

A. I and III only
B. II and III only
C. II and IV only
D. I, II and IV only
E. I, III and IV only

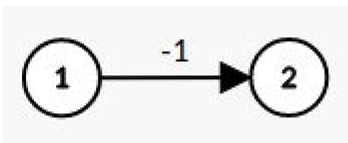
Refer to Lecture 7 slides. Both insertion sort and selection sort can be simulated by rotating
the elements in a queue.

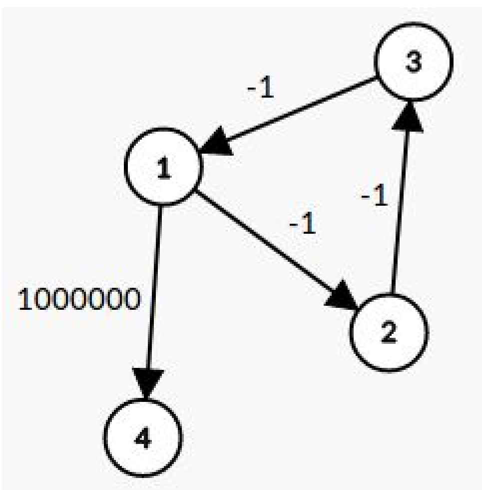
Section 2: Analysis Questions (15 marks)

Prove (the statement is correct) or disprove (the statement is wrong) the statements below.

11a. Suppose we have a graph with *n* vertices and *m* edges. First run the Bellman-Ford Algorithm on the graph from source *s*. Then, relax all edges for a further n^2 iterations. Only all vertices *v* where dist(*s*, *v*) < 0 (according to the algorithm above) have $-\infty$ shortest distance. (3 marks)

False. Counterexamples:





11b. Suppose we have a undirected graph with non-negative edge weights. If we run the Modified Dijkstra's Algorithm (mentioned in lecture) with a max Priority Queue, we cannot solve the Single-Source Shortest Path problem. (3 marks)

False. Relax function guarantees the correctness of finding the shortest distance to a node.

When the shortest distance to a node is found, it will never be added back to the Priority Queue again. Number of unique nodes in Priority Queue shrinks over time (despite the ordering), hence Dijkstra's Algorithm with a max Priority Queue will eventually terminate.

11c. The following hash function is **not** a good hash function for strings. (3 marks)

```
int hash(String s) {
    int a = 11, h = 0;
    for (int i = 0; i < s.length(); i++) {
        h %= a;
        h *= ((int)s.charAt(i) + 11);
        a += 2;
    }
    return h + a;
}
```

True. Nothing will be hashed to slots 0 to 10, and hash function hashes based on the length of the input string, and hence may result in a large number of collisions.

11d. In each iteration of radix sort, we order the data into groups according to the next character in each data. Insertion sort can be used to perform this ordering. (3 marks)

True. A stable sorting algorithm is required, and insertion sort is a stable sorting algorithm.

11e. In a max binary heap data structure, we are able to find the successor of any element in constant time. (3 marks)

False. O(*n*) time is required.

Section 3: Application Questions (55 marks)

12a) In a city of *n* towns, *m* directed roads connect two towns. Each road has a positive height limit, where only vehicles with height lower than the height limit can travel on and pass through the road. Design an algorithm that allows you to find, for every pair of towns, the height of the highest vehicle that can travel between them. (5 marks)
Hint: The Floyd Warshall's Algorithm taught in class is provided for you below.

```
Let the number of nodes in the graph be V.
Let D be an adjacency matrix.
Let D[i][i] = 0, D[i][j] = the weight of edge(i, j) if there is an
edge from i to j, otherwise ∞.

for (int k = 0; k < V; k++)
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            D[i][j] = Math.min(D[i][j], D[i][k] + D[k][j]);
```

Modify Floyd Warshall's Algorithm as follows (changes in red):

```
Let the number of nodes in the graph be V.
Let D be an adjacency matrix.
Let D[i][i] = 0, D[i][j] = the weight of edge(i, j) if there is an
edge from i to j, otherwise 0.

for (int k = 0; k < V; k++)
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            D[i][j] = Math.max(D[i][j], Math.min(D[i][k], D[k][j]));
```

12b) In a city of n towns, m directed roads connect two towns. There are two types of roads: normal roads and toll roads. Coco's home is located at node 1 and he wants to get to school, located at node n. Describe the most efficient algorithm you can think of to help him find the minimum number of toll roads he has to pass through on his way from home to school. (10 marks)

Model towns as nodes, roads as weighted edges - weight 1 if it is a toll road, and weight 0 otherwise.

Run Dijkstra's Algorithm from node 1, and we can obtain the shortest distance from node 1 to node n. This runs in $O((n + m) \log n)$. However, there is a more efficient solution.

Observe that the Priority Queue in Dijkstra's Algorithm ensures that nodes in the Priority Queue are ordered by increasing distance from node 1, such that nodes closer to node 1 are always processed first. If we can enforce this ordering without a Priority Queue, we may be able to achieve a better runtime.

It turns out that we can substitute the Priority Queue with a double-ended queue. To enforce the ordering, when we enqueue the neighbour of a node into the double-ended queue, if the neighbour can be reached by passing through an edge with weight 0, then enqueue the neighbour to the front of the double-ended queue. Otherwise, enqueue the neighbour to the back of the double-ended queue. Again, this will always ensure that the nodes in the queue are ordered in increasing distance from the source node (node 1), and hence ensures that the shortest distance is found correctly. This algorithm runs in $O(n + m)$.

12c) In a city of *n* towns, *m* directed roads connect two towns. Roads are of positive lengths, some are long, some are short, and two roads may be of the same length. Coco's home is located at node 1 and he wants to get to school, located at node *n*, via a simple path. The *boringness* of a path is defined as the *k*th longest road on that path. Describe an algorithm to find the *boringness* of the least boring path among all paths Coco can take. (15 marks)

Rephrase definition of *boringness* of a path: The *boringness* of a path is *w* if there are *k* - 1 edges on the path with weight greater than *w*.

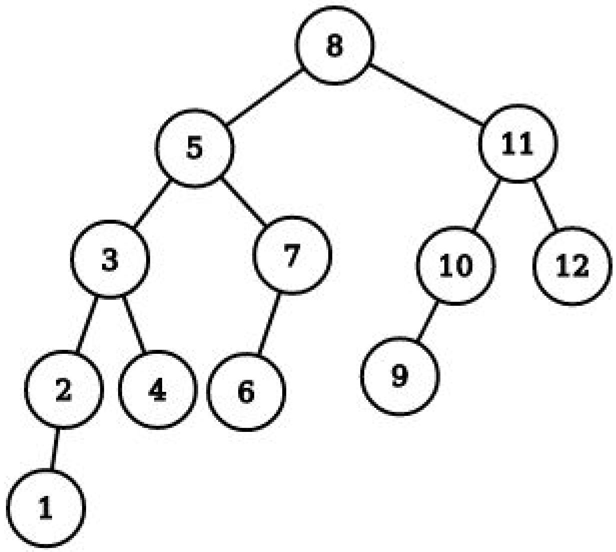
Transform problem: Given a value of *w*, what is the minimum number of edges on a path between node 1 and *n* with weight greater than *w*.

We can model the transformed problem as a shortest path problem. Imagine edges with weight > *w* are toll roads, and edges with weight < *w* are normal roads. Then this problem is exactly the same as the problem in Q12b. We can then relabel edges with weight > *w* with weight 1, and edges with weight < *w* with weight 0. Then, we run the algorithm in Q12b, and we are able to obtain the minimum number of edges with weight > *w* that we need to pass through to travel from node 1 to node *n*.

Now, for a given value of *w*, we can compute the minimum number of edges required. We now want to find the value of *w* that makes the minimum number of edges required to be exactly *k*. There are two ways to do this:

- Run the algorithm above for all unique edge weight values in the graph, and output the one that gives *k* as the answer - $O(n + m^2)$
- Binary search for the value of *w* that gives *k* as the answer - $O((n + m) \log m)$

12d) Given the following AVL tree, show what happens when node 11 is deleted. (10 marks)



Solution is not provided for this question.

12e) Given a sorted array of *n* integers, write a method `buildAVL` that builds a height balanced AVL tree in O(*n*). (15 marks)

Set the median element of the array as the root of the AVL tree. Then, recursively divide the array into two halves, and set the median of each of the two halves to be the children of the current node.

End of Paper