THE UNIVERSITY OF ADELAIDE
AUSTRALIA

Primary Examination, Semester 2, 2017

Algorithm and Data Structure Analysis
COMPSCI 2201, 7201

Writing Time: 120 mins

Questions	Time	Marks
Answer all 7 questions	120 mins	120 marks
		120 Total

Instructions

- Begin each answer on a new page
- Examination material must not be removed from the examination room
- Only Simple Calculators Allowed

Materials

- 1 Blue book
- Textbooks and slides (all paper based permitted)
- 1 Dictionary for translation purposes only

DO NOT COMMENCE WRITING UNTIL INSTRUCTED TO DO SO

Right or Wrong?

Question 1

(a) Indicate whether each of the following statements is true or false. There is one mark for each correct answer and zero marks for each incorrect answer.

	Statement	
1	$log(n)^{10} \in \Omega(n)$ False	
2	$2.1 \cdot n^{2.1} + 1500n^2 \in \Theta(n^3)$ False	
3	$3n^2 + 1 \in o(n^2)$ False	
4	Insertion (assuming the added pair does not already exist) for hash table (with chaining) takes $O(1)$ time in the worst case	True
5	It is known that there are problems in NP that are not in P	True
6	The problem to decide whether a given graph contains an Eulerian cycle is NP-hard	??
7	The Bellman-Ford single-source shortest path algorithm can work on graphs containing negative-weight cycles.	True
8	AVL trees always have $O(\log n)$ time for *delete* operations in the worst case	true
9	The Dijkstra single-source shortest path algorithm cannot work on graphs containing negative-weight cycles.	true
10	Hash-tables can maintain $O(1)$ access times in the worst case.	False

[10 marks]

[Total for Question 1: 10 marks]

Proofs and Sequences

Question 2

(a) Write down the pseudocode for Katratsuba multiplication.

[4 marks]

(b) Prove by induction that the sum:

$$2^0 + 2^1 + 2^2 + \ldots + 2^n = 2^{n+1} - 1$$

[7 marks]

(c) Solve the following recurrence using the Master Theorem. Briefly explain your solution.

$T(n) = 4T(\frac{n}{2}) + n^2$

[5 marks]

[Total for Question 2: 16 marks]

2 a) function karatsuba (a, b, n) {

```
if n < 4
    return a x b
else
    a1, a0 = split (a)
    b1, b0 = split (b)

    k = ceil (n/2)

    p0 = karatsuba( a0, b0, k)
    p1 = karatsuba ( a1, b1, k)
    p2 = karatsuba ( [a0 + a1], [b0 + b1] , k) - p0 - p1

    return b^2k · p2 + b^k · p1 + p0
```

b) Base Case: Prove true for

$n = 0$

$$2^0 = 2^{0+1} - 1$$
$$1 = 2 - 1$$
$$= 1$$

$\therefore$ true for $n = 0$

Inductive Hypothesis: Assume true for $n = k$

$$2^0 + 2^1 + 2^2 + \ldots + 2^k = \underline{2^{k+1} - 1}$$

Inductive Step: Prove true for $n = k+1$

$$\underline{2^0 + 2^1 + 2^2 + \ldots + 2^k} + 2^{k+1} = 2^{(k+1)+1} - 1$$

$$2^{k+1} - 1 + 2^{k+1} = 2^{(k+1)+1} - 1$$

$$4^{k+1} - 1 = 2^{k+1} \cdot 2 - 1$$
$$= 4^{k+1} - 1$$

LHS = RHS Q.E.D

2 c) $a = 4$

$b = 2$

$d = 2$

$a = b^d$

$4 = 2^2$

$T(n) \in O(n^2 \log n)$

The work is distributed evenly across each level of recursion

Hashing and Skiplists

Question 3

(a) A hash function, $h(x)$, hashes a name, x (the hash key), onto a hash value (the index into an array) as listed in the following table. The hash table has a size of 8 (indexed from 0 to 7).

The keys are inserted into a hash table in the order listed in the following table. Use the following collision resolution approaches as described in the lectures/tutorials.

y	h(y)
Lime	6
Mandarin	5
Mango	5
Grapefruit	1

Show the hash table contents after insertion with:

i. chaining

[2 marks]

ii. linear probing

[2 marks]

~~iii. quadratic probing~~

[2 marks]

iv. For a hash table based on linear probing with n elements, what is the worst-case complexity for deleting an element.

[3 marks]

(b) Derive the expression for the expected height H of an element inserted into a skiplist.

[3 marks]

(c) Given a specific list of keys, $h_1(key)$ maps the keys uniformly to values from 0 to 13. $h_2(key)$ maps the keys uniformly to values from 0 to 25. Now I want to build a hash table of size around 39 (still only dealing with the original set of keys). Is using $h(key) = h_1(key) + h_2(key)$ a good idea? Why or why not?

[2 marks]

[Total for Question 3: 14 marks]

3a) i)

1. insert lime

0	1	2	3	4	5	6	7
						lime	

2. insert mandarin

0	1	2	3	4	5	6	7
					mandarin	lime	

3. insert mango

0	1	2	3	4	5	6	7
					mandarin → mango	lime	

4. insert grapefruit

0	1	2	3	4	5	6	7
	grapefruit				mandarin → mango	lime	

ii).

1. insert lime

0	1	2	3	4	5	6	7
						lime	

2. insert mandarin

0	1	2	3	4	5	6	7
					mandarin	lime	

3. insert mango

0	1	2	3	4	5	6	7
					mandarin	lime	mango

4. insert grapefruit

0	1	2	3	4	5	6	7
	grapefruit				mandarin	lime	mango

iv). $O(n)$ – in the worst case, we must traverse the entire table to find the key

Trees

Question 4

(a) Draw a maximally balanced binary search tree that can be produced from the elements: $1, 2, 3, 4, 5, 6, 7, 8, 9$. *Hint:* a maximally balanced binary search tree minimises the average depth of its elements.

[4 marks]

(b) Draw a sequence of diagrams showing the insertion of the values:

`[1,9,8,7,2,3,6,4,5]`

into an empty AVL tree, in the order shown above.

You must:

- Show the resulting tree immediately after each insertion step (that is *before* any balancing has taken place).
- Show the resulting tree after balancing operation(s).

[10 marks]

(c) What is the average cost of find on a binary search tree generated by the insertion of the elements: 10, 1, 6, 9, 7, 3, 2, 8, 4, 5 into an empty tree. Show your working.

[6 marks]

(d) What is the complexity for searching/adding/removing from an AVL tree with n nodes.

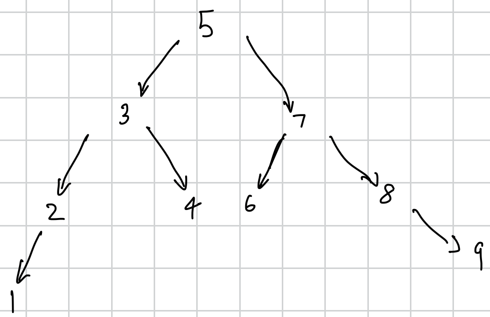
[3 marks]

(e) State two properties of a binary search tree?

[2 marks]

[Total for Question 4: 25 marks]

4 a).

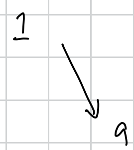


b) 1. insert 1

1

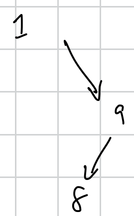
2. no balancing

3. insert 9



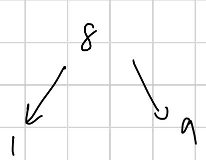
4. no balancing

5. insert 8

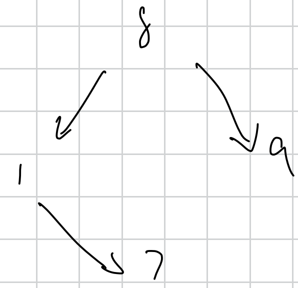


6. right rotation on 9

7. left rotation on 8



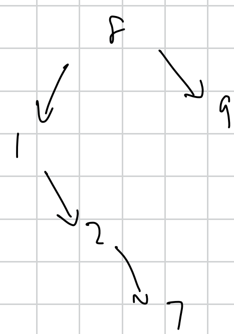
8. insert 7



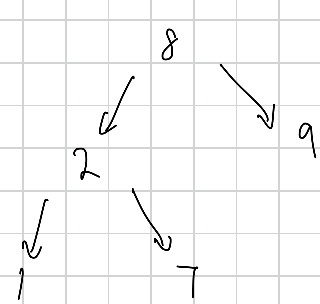
9. no balancing

10. insert 2

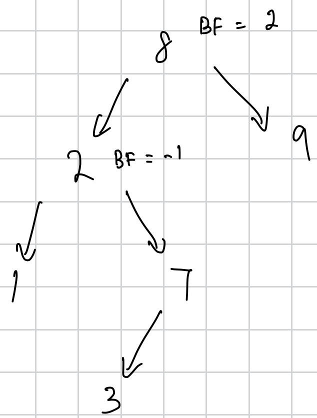
11. right rotation on 7



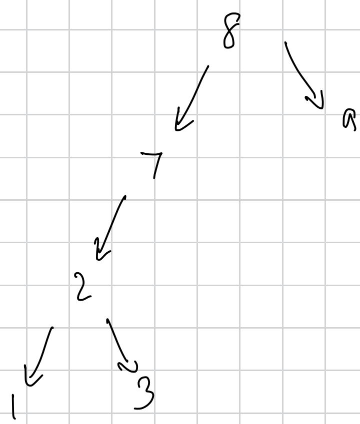
12. left rotation on 1



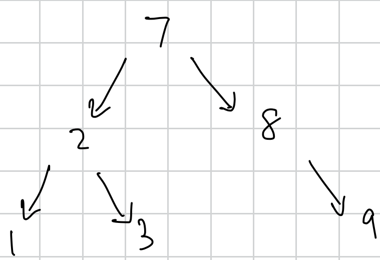
13. insert 3



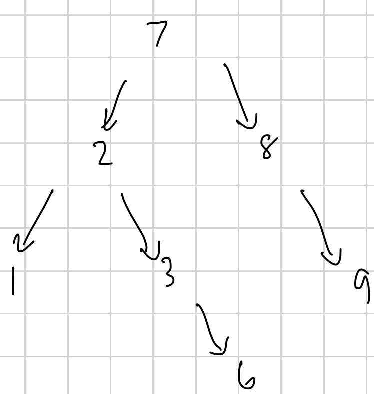
14. left rotation at 2



15. right rotation at 8

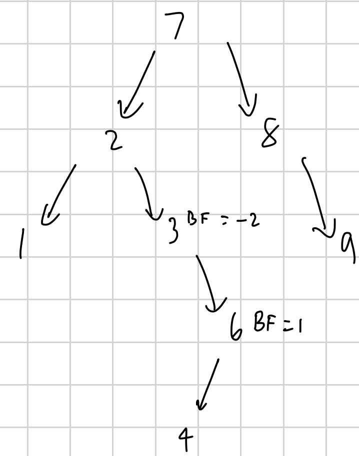


16. insert 6

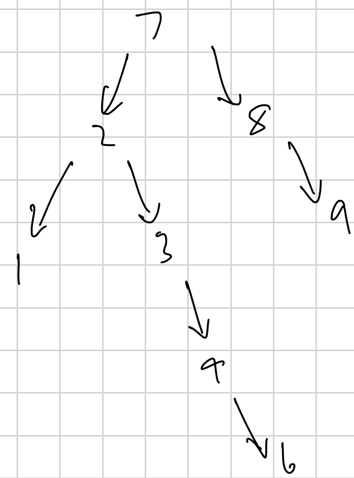


17. no balancing

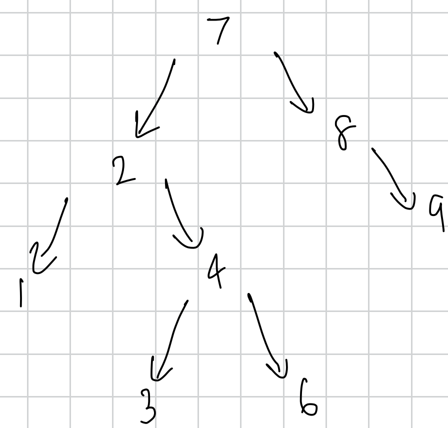
18. insert 4



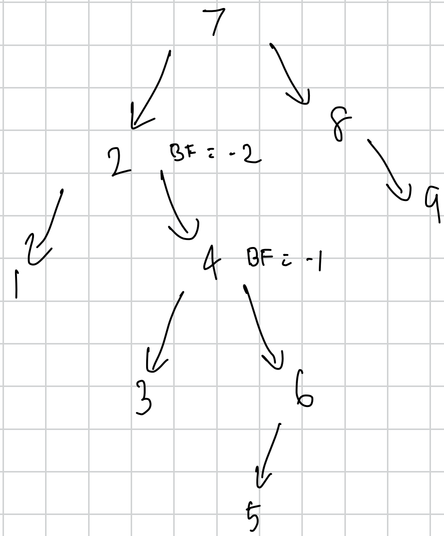
19. right rotation at 6



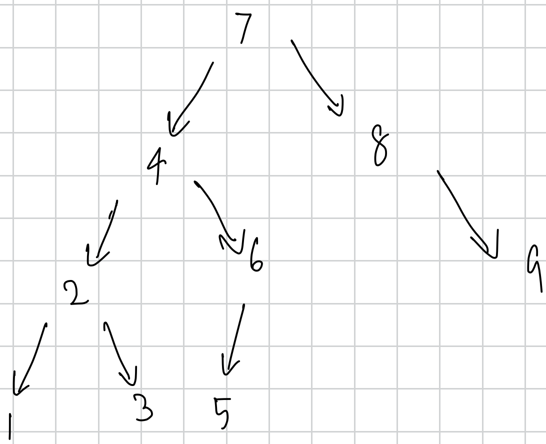
20. left rotation at 3



21. insert 5

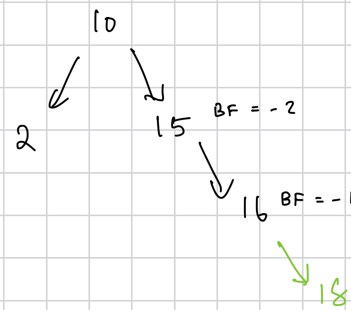


22. left rotation at 2

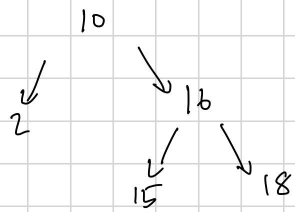


Final !!

Exercise : insert (18)



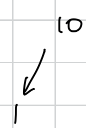
left rotation at 15



c) 1. insert 10

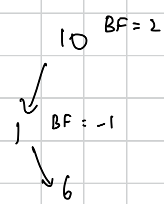
10 cost = 1

2. insert 1



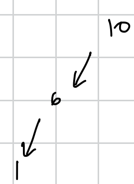
cost = 2

3. insert 6

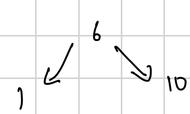


cost = 3

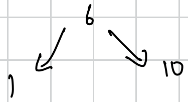
left rotation at 1

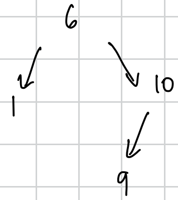


right rotation at 10



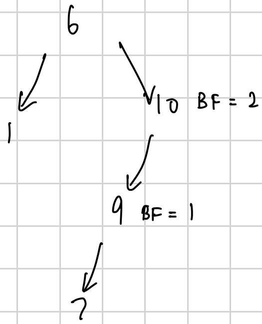
4. insert 9





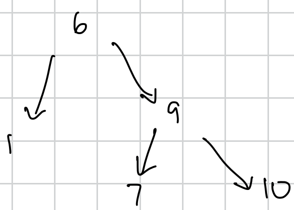
cost = 3

5. insert 7

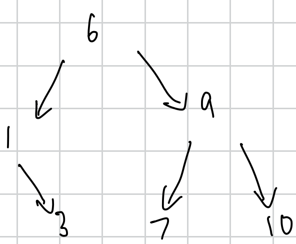


cost = 4

right rotation at 10

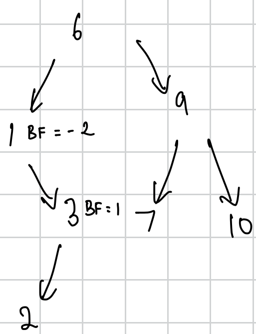


6. insert 3



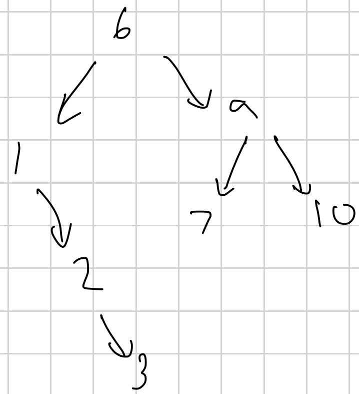
cost = 3

7. insert 2

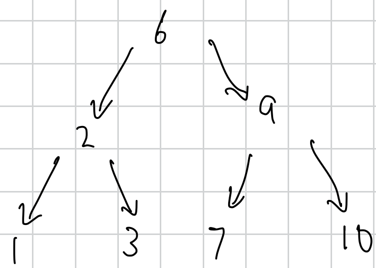


cost = 4

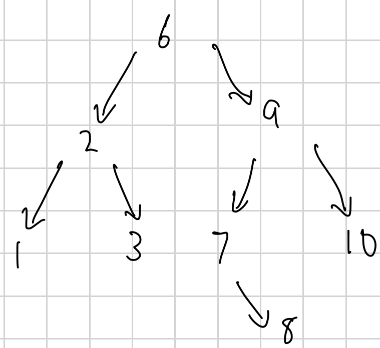
right rotation at 3



left rotation at 1

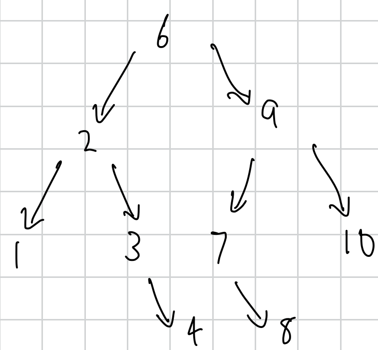


8. insert 8



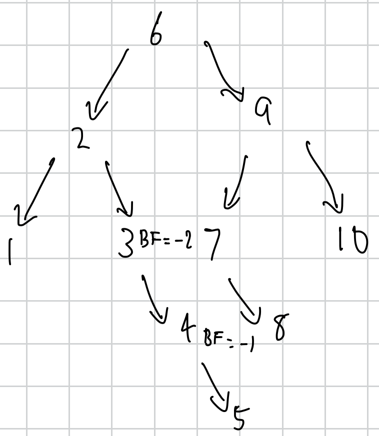
cost = 4

9. insert 4



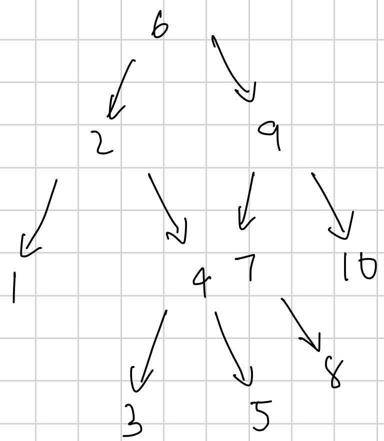
cost = 4

10. insert 5



cost = 5

left rotation at 3



$$\text{average cost} = \frac{1+2+3+3+4+3+4+4+4+5}{10}$$

$$= \frac{33}{10}$$

$$= 3.3$$

d) $O(\log n)$. since AVL trees maintain a height of $\log(n)$, it ensures that find, delete and insert all operate in $\log n$ at worst.

e)
- for every node, all keys in left subtree of x are smaller than x, and all keys in the right subtree of x, are bigger than x
- all nodes in a BST are unique
- if each node has a balance factor $\in \{-1, 0, 1\}$, then the BST is said to be perfectly balanced and has a height of $\log n$

Graph Representations and Traversals

Question 5

(a) Write pseudo-code for an algorithm that performs depth-first search of a directed graph: $G = (V, E)$ and prints out all of the **nodes** as they are *checked*. Note: a node is *checked* when it is inspected by the algorithm for the first time.

[9 marks]

(b) State the storage requirements of a graph with n nodes and m edges using:

1. an adjacency list, and
2. an adjacency matrix

briefly justify each answer.

[4 marks]

(c) Given a directed graph $G = (V, E)$. We know that the edges have nonnegative costs. Also, there are a lot of edges with 0 costs. Given a source node s, we need to find out which nodes have 0 distance from s. Propose an algorithm for this problem.

[5 marks]

[Total for Question 5: 18 marks]

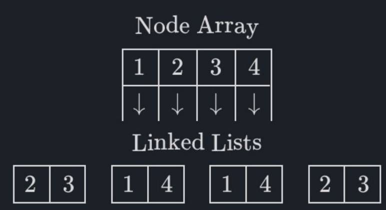
5a)

```
function DFS(graph, startNode):
    stack = new Stack()
    visited = new
Array(graph.numNodes).fill(false)

    stack.push(startNode)

    while stack is not empty:
        currentNode = stack.pop()

        if not visited[currentNode]:
            visited[currentNode] = true

            for each neighbor of currentNode:
                if not visited[neighbor]:
                    stack.push(neighbor)

    return visited
```

b)

Adjacency Lists

Adjacency lists store nodes inside a node array. Each element inside this array points to a linked list which contains the edges for that node.



Advantages: $o(m)$, $m =$ # edges

- Edge insertion and deletion can be done in $O(1)$ time.
- Memory usage is proportional to the number of edges, making it efficient for sparse graphs.

Disadvantages:

- Edge queries (checking if an edge exists) require traversing the linked lists, taking $O(n)$ time in the worst case, where n is the number of vertices.
- Not as cache-friendly as adjacency matrices due to the non-contiguous memory layout of linked lists.

Adjacency lists strike a balance between space efficiency and the ability to modify the graph structure dynamically, making them a popular choice for many graph algorithms.

Adjacency Matrices

An adjacency matrix is a 2D array that represents a graph's connections. For a graph with n vertices, the matrix has dimensions $n \times n$. The entry at row i and column j is 1 if there is an edge from vertex i to vertex j, and 0 otherwise.

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 0 \\ 2 & 1 & 0 & 0 & 1 \\ 3 & 1 & 0 & 0 & 1 \\ 4 & 0 & 1 & 1 & 0 \end{array}$$

$O(n^2)$, n = # nodes.

Advantages:

- Edge queries can be done in $O(1)$ time by checking the corresponding matrix entry.

Disadvantages:

- Space complexity is $O(n^2)$, even for sparse graphs with few edges.
- Adding or removing vertices requires resizing the matrix, which can be costly.

Adjacency matrices are preferred when the graph is dense (many edges) and fast edge queries are crucial. However, for large, sparse graphs, the space overhead can be significant.

c) Refool Dijkstra - Since there are no negative edges, a distance of 0 to a node indicates shortest path.

Therefore we just find the shortest path to every node, traverse this path and count the distance. If = 0, then node is 0 distance from starting node.

Shortest Path Algorithms

Question 6

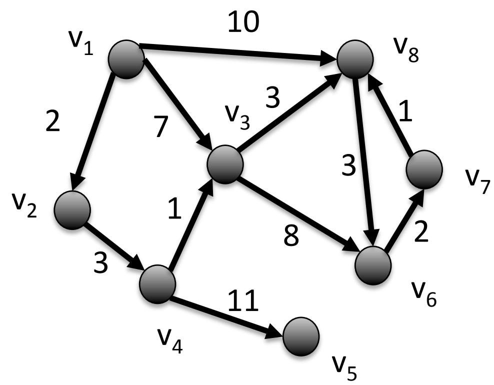
(a) Breadth-first search (BFS) can be used to perform single source shortest paths on any graph where all edges have the same costs 1. Write an algorithm using BFS to assign distances to all nodes in a graph.

[7 marks]

(b) If we change the setting of part (a) this way: all edges have costs 1 except for a constant number of edges, which all have costs 2. Can you still use BFS to assign distances?

[3 marks]

(c) Consider the following graph:



Solve the single-source-shortest path problem for the start node v_1 using Dijkstra's algorithm. List for each iteration which nodes becomes scanned.

[9 marks]

(d) Briefly explain, with the use of an example, why Dijkstra's algorithm will not work on graphs with negative weight edges.

[4 marks]

[Total for Question 6: 23 marks]

6 a)

```
function BFS(graph, startNode):
    queue = new Queue()
    visited = new Array(graph.numNodes).fill(false)
    distance = new Array(graph.numNodes).fill(infinity)
    parent = new Array(graph.numNodes).fill(null)

    queue.enqueue(startNode)
    visited[startNode] = true
    distance[startNode] = 0

    while queue is not empty:
        currentNode = queue.dequeue()

        for each neighbor of currentNode:
            if not visited[neighbor]:
                queue.enqueue(neighbor)
                visited[neighbor] = true
                distance[neighbor] = distance[currentNode] + 1
                parent[neighbor] = currentNode

    return (visited, distance, parent)
```

b) No! BFS works because it assumes every possible path is the shortest, and only checks each node once. If the first path checked is not the shortest, it will incorrectly identify that path as the shortest (because of how the queue works)

c)

Step 1: initial node

Visited : $\{V_1\}$
Unvisited : $\{V_2, V_3, V_4, V_5, V_6, V_7, V_8\}$

nodes :	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
distance :	0	2	7	∞	∞	∞	∞	10
parent :	/	V_1	V_1	/	/	/	/	V_1

Step 2: next shortest node = V_2

Visited : $\{V_1, V_2\}$

Unvisited : $\{V_3, V_4, V_5, V_6, V_7, V_8\}$

nodes :	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
distance :	0	2	7	5	∞	∞	∞	10
parent :	/	V_1	V_1	V_2	/	/	/	V_1

Step 3: next shortest node = V_4

Visited : $\{V_1, V_2, V_4\}$

Unvisited: $\{V_3, V_5, V_6, V_7, V_8\}$

nodes :	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
distance :	0	2	6	5	11	∞	∞	10
parent :	/	V_1	V_4	V_2	V_4	/	/	V_1

Step 4: next shortest node = V_3

Visited : $\{V_1, V_2, V_4, V_3\}$

Unvisited: $\{V_5, V_6, V_7, V_8\}$

nodes :	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
distance :	0	2	6	5	11	15	∞	10
parent :	/	V_1	V_4	V_2	V_4	V_3	/	V_1

Step 5: next shortest node = V_8

Visited : $\{V_1, V_2, V_4, V_3, V_8\}$

Unvisited: $\{V_5, V_6, V_7\}$

nodes :	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
distance :	0	2	6	5	11	13	∞	10
parent :	/	V_1	V_4	V_2	V_4	V_8	/	V_1

Step 6: next shortest node $= V_5$

visited : $\{V_1, V_2, V_4, V_3, V_8, V_5\}$

unvisited: $\{V_6, V_7\}$

nodes :	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
distance :	0	2	6	5	11	13	∞	10
parent :	/	V_1	V_4	V_2	V_4	V_8	/	V_1

Step 7: next shortest node $= V_6$

visited : $\{V_1, V_2, V_4, V_3, V_8, V_5, V_6\}$

unvisited: $\{V_7\}$

nodes :	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
distance :	0	2	6	5	11	13	15	10
parent :	/	V_1	V_4	V_2	V_4	V_8	V_6	V_1

Step 8: next shortest node $= V_7$

visited : $\{V_1, V_2, V_4, V_3, V_8, V_5, V_6, V_7\}$

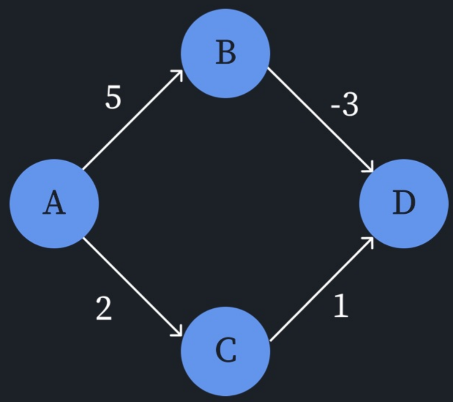
unvisited: $\{\}$

nodes :	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
distance :	0	2	6	5	11	13	15	10
parent :	/	V_1	V_4	V_2	V_4	V_8	V_6	V_1

d)

Dijkstra's Algorithm and Negative Edge Weights

Consider the following graph:



If we run Dijkstra's algorithm starting from node A, it will first visit node C with a distance of 2. Then, it will visit node D with a distance of 3 (via the path A -> C -> D). At this point, node B is still in the queue with a distance of 5.

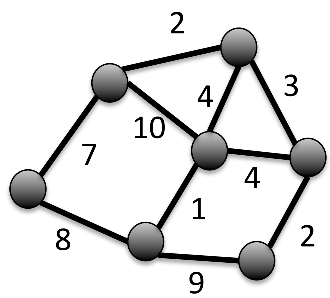
When the algorithm dequeues node B, it will find that the distance to D from B is 2 (5 - 3). However, Dijkstra's algorithm will not update the distance to D because it has already been visited and marked as final.

The actual shortest path from A to D is A -> B -> D with a total distance of 2. Dijkstra's algorithm misses this path because it greedily marks the distances as final and does not revisit nodes to update their distances when a negative edge is encountered.

Minimum Spanning Trees and P vs NP

Question 7

(a) Draw two different minimum spanning trees for the graph below.



In your answer show the final trees including the weights on the links of the trees.

[6 marks]

(b) Write down the pseudocode of the Jarnik Prim algorithm.

[4 marks]

(c) Prove that the minimum spanning tree problem is in P.

[4 marks]

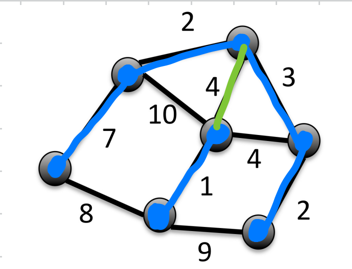
[Total for Question 7: 14 marks]

End of exam

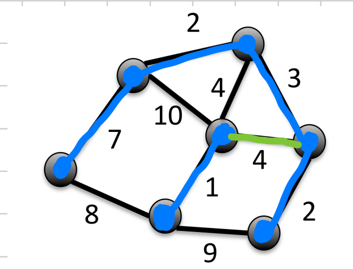
7a) Using kruskal

edges sorted: 1, 2, 2, 3, 4, 4, 7, 8, 9, 10

diverges with choice of 4



OR



b).

```
function PrimMST(Graph, startingNode):
    visited = {startingNode}
    mst = []
    pq = priorityQueue()

    for each edge(startingNode, v) in Graph:
        pq.enqueue(edge(startingNode, v))

    while not pq.isEmpty():
        minEdge = pq.dequeue()
        if minEdge.to not in visited:
            visited.add(minEdge.to)
            mst.append(minEdge)
            for each edge(minEdge.to, v) in Graph:
                if v not in visited:
                    pq.enqueue(edge(minEdge.to, v))

    return mst
```

c). A problem in P can be solved in deterministic Polynomial time.

Assuming a graph is denoted using $G = (E, V)$, where E is edges, V is nodes,

then we know that an MST can be found using kruskal's algrothim in $O(E \log E)$ time. $O(E \log E) \in P$. Therefore MST problem is in P. Q.E.D.