Student Number (Required) ______________________

SOLUTIONS

Name (Optional) ________________________________

This is a closed book test. You may not refer to any resources.

This is a 50 minute test.

Please write your answers in ink. Pencil answers will be marked, but will not be re-marked under any circumstances.

The test will be marked out of 50.

<table>
<thead>
<tr>
<th>Question 1</th>
<th>/10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 2</td>
<td>/10</td>
</tr>
<tr>
<td>Question 3</td>
<td>/10</td>
</tr>
<tr>
<td>Question 4</td>
<td>/10</td>
</tr>
<tr>
<td>Question 5</td>
<td>/10</td>
</tr>
<tr>
<td>TOTAL</td>
<td>/50</td>
</tr>
</tbody>
</table>
General marking philosophy: a student who gives enough of an answer to show they understood what they were supposed to do, even if they couldn’t do it (or made lots of errors while doing it) should get about 50% on that question.

Full marks should be given if a solution is sound and not missing anything important.

Many of these questions can be answered in different ways – for example, there are other algorithms that could be used for Questions 3 and 4. It is not required that the students use the same algorithms as I do.

A student should only get 0/10 on a question if they made no attempt to answer it at all.
Question 1 (10 marks)

Write a function that searches a binary search tree T for a specific value x. If x is in the tree, the function returns the depth at which the value was found. If x is not in the tree, the function returns -1. You may assume that the depth of the root is 0.

Your function can be recursive or iterative, as you choose.

Solutions:

Recursive:

def find_depth(T, x):
    return rfd(T.root, x)

def rfd(v, x):
    if v == nil:
        return -1
    else:
        if v.value == x:
            return 0
        else:
            if v.value > x:
                t = rfd(v.left, x)
            else:
                t = rfd(v.right, x)
            if t == -1:
                return -1
            else:
                return t+1
Iterative:

```python
def find_depth(T, x):
    current = T.root
    d = 0
    while current != nil:
        if current.value == x:
            return d
        else:
            d += 1
            if current.value > x:
                current = current.left
            else:
                current = current.right
    return -1
```

Marking:

As explained above, a student who understood the question and had a good idea of how to answer it should get at least 5/10. Beyond that, take one or two marks off for errors of logic or or missed cases (for example, if the solution does not return -1 properly when the search value is not in the tree, take 2 marks off; if the returned value is always off by 1, take 1 mark off).
Question 2 (10 marks)

Here is a binary tree. The square vertices represent empty leaf-nodes. Either explain why this tree cannot be legally coloured as a Red-Black Tree, or give it a legal Red-Black colouring.
Solution:

Consider the left side of the tree. X can be either Red or Black. If X is Red, then the path from X down to the left-most leaf contains 1 Black vertex. Since X is Red, Y must be Black, so the path from X down to either leaf below Z will contain at least 2 Black vertices. This is illegal. Similarly if X is Black, the path from X down to the left-most leaf contains 2 Black vertices. Y and Z cannot both be Red, so the path from X down to either leaf below Z will contain at least 3 Black vertices. This is illegal.

Thus X cannot legally be coloured either Red or Black. Thus this tree cannot be legally coloured.

Marking:

Students may come up with other arguments, but they need to show that any attempted colouring will break at least one of the rules.

Take 1 or 2 marks off if their answer is incomplete (for example, claims that a path must contain 3 black vertices without mentioning that this is due to the Red-Red prohibition)

If a student claims that the tree can be coloured and gives an invalid colouring to demonstrate this, give 3/10
Question 3 (10 marks)

What is the big-O (worst case) complexity of the total time required to build a Red-Black tree containing n values (i.e. what is the order of the total time required to perform n insertions, starting with an empty tree)? Explain your answer.

Solution:

We know the number of levels in the RB tree is $O(\log n)$. Each insertion may require searching to the very bottom of the tree ($O(\log n)$ time), attaching a new vertex ($O(1)$ time) then $O(1)$ time if we rotate and exit, $O(\log n)$ time if we rotate and recurse back up the tree.

Thus each insertion takes $O(\log n)$ time. Since there are n insertions, the total complexity is $O(n \times \log n)$

Marking:

The most common error here will most likely be to give $O(\log n)$ as the answer – this is the complexity of a single insertion, but for this question we need to make n insertions. For this answer, give 6/10

If they give $O(n)$ or $O(n^2)$ give 4/10
Question 4 (10 marks)

Here is a Red-Black Tree. Show the tree that results after the value 12 is inserted into this tree.

In this drawing ○ signifies a Red vertex

You may draw your solution on the next page.
Solution: I am drawing the solution in three stages – the students are only required to draw the final condition of the tree.

Step 1: Add the new value:

Step 2: Fix the immediate Red-Red Problem by recolouring
Step 3: Fix the new Red-Red Problem with a double rotation (which restores the balance)

Marking:

If the student makes a mistake during the rotation but still comes up with a valid RB tree, give about 7/10. If they end up with a tree that is not a valid RB tree but it is clear that they understand that the tree has to rotate, give 5/10.

As noted above, they do not have to show the stages of the process, just the end result.
Question 5  (10 marks)

Consider a hash table of size 7 with hash function \( h(k) = k \mod 7 \). Draw the table that results after inserting the values 19, 26, 17, 3, 18, in that order …

a) when collisions are resolved by chaining [2 marks]

b) when collisions are resolved by linear probing [4 marks]

c) when collisions are resolved by quadratic probing using \( c_1 = 1 \) and \( c_2 = 1 \)
Solutions:

a)
b) is worth 2 marks. The two possible answers are the result of adding collided values to the head of each list, or to the tail of each list. Either is fine but the student should be consistent. Give 1 mark if the student understands the concept of chaining but gets it wrong.

b) and c) are each worth 4 marks. Give 2 if the student understands the collision resolution method but gets it totally wrong. Give 3 if they know what to do but make a mistake at some point.
Nearly Blank Page With Some Helpful Info:

<table>
<thead>
<tr>
<th>$k$</th>
<th>$k \mod 7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>26</td>
<td>5</td>
</tr>
</tbody>
</table>