CS2515

2023/24

CLASS TEST 2

ANNOTATED SOLUTIONS

29/11/23

You have 45 minutes to complete answers to these questions.

Please complete the information on this page.

There are questions worth 30 marks on the paper. You should answer all questions.

Write your answers directly into the spaces on the question sheet.

The marks you receive will be scaled, and will contribute up to 10 marks towards the final total for CS2515.

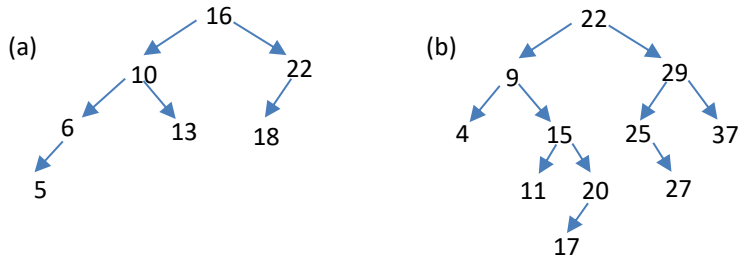
Student ID number:

Place X in box if registered with DSS: ☐

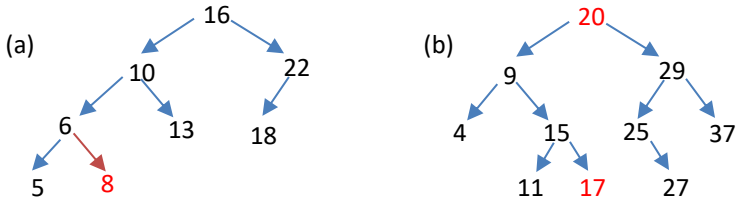
Place X in box if availing of a spelling and grammar waiver: ☐

(for marking guidelines, see https://www.ucc.ie/en/dss/publications/)

1. For the *Binary Search Trees* below, show the result when you add key 8 to the tree in (a), and the result when you remove key 22 from the tree in (b).



(5 marks)



(For (b), also accepted moving 25 into the root, and then 27 into 25's place)

2. We can represent a node in a Binary Search Tree as an object instance of a class BSTNode. State the fields (ie variables) of BSTNode, and then give clear recursive pseudocode for searching for an element x in the tree rooted at a node, and returning the element if found. Use the fields you specified for the BSTNode.

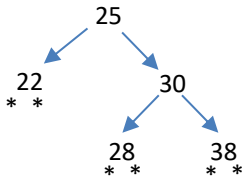
(5 marks)

Fields: parent, leftchild, rightchild, element

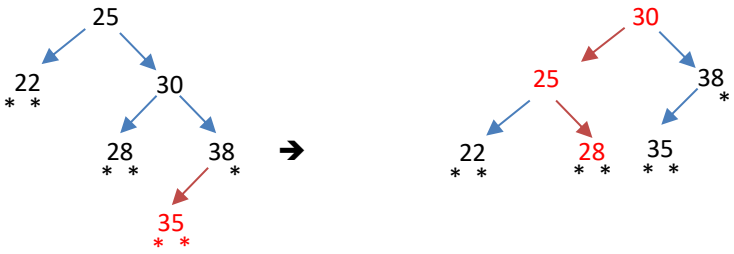
```
search(self, item):
    if self.element == item
        return self.element
    elif self.element < item
        if self.leftchild exists
            return self.leftchild.search(item)
    else
        if self.righthchild exists
            return self.rightchild.search(item)
    return None
```

Must be recursive. Must state the fields. Final return is to catch cases where node does not have appropriate child; could have written as standalone recursive method, passing 'node' in as first argument. Every path through the method must end in a return statement. Note that the node is not the element – you can't compare node against item. Full python code was also accepted instead of pseudocode.

3. For the AVL tree below, show the result of adding 35.

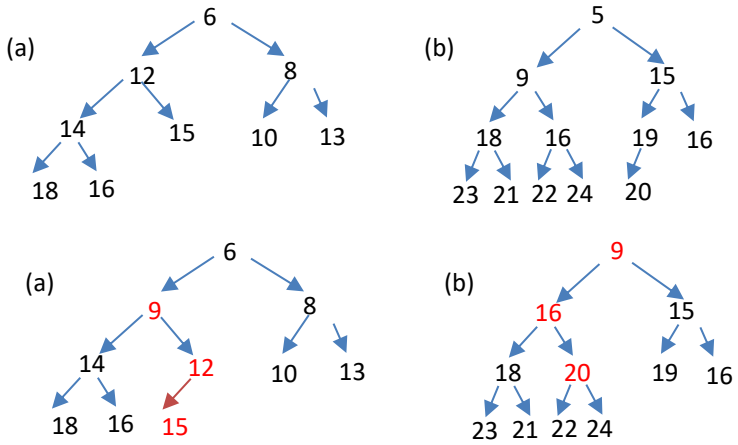


(4 marks)



After adding 35, and computing the heights, 25 is the unbalanced node (leftchild has height 0, rightchild has height 2). Height is caused by a straight line, so single rotation of 30 into 25, with 28 swapping sides.

4. The two trees below represent *min Binary Heaps* (and so they are **not** Binary Search Trees). Show the results when you add 9 to the min binary heap in (a), and you remove 5 from the min binary heap in (b):



(6 marks)

(a)New item added left of 15, then bubbled up to correct place.
(b) 'last' item 20 moved into root, then bubbled down (swap with smaller child) until correct place.

5. In terms of *n*, the number of items in a (min) Binary Heap implemented using a python (array-based) list, what is the worst case complexity of

(a) reading the highest priority (lowest key) item? **O(1)**

(b) adding a new (key,element) item to the heap? **O (log n)***

(c) removing the highest priority (lowest key) item from the heap? **O(log n)***

Possible to argue individual worst case for (b) and (c) should be O(n), so that was also accepted.

(3 marks)

6. We are given the following hash function, represented as a lookup table:

input:	351	352	353	354	355	356	357	358
hash:	4628	7649	5491	6529	2898	7385	4206	7918

If we start with a hash table implemented using an empty array of size 10, where the table is using open addressing with linear probing, with standard mod compression, show the changes to the array as we execute each of the following steps in turn, using the above hash function. There is no need to grow the array.

Steps	0	1	2	3	4	5	6	7	8	9
setitem(354,'P'')										(354,'P')
setitem(358,'Q')									(358,'Q')	|
setitem(355,'R')	(355,'R')								|	|
setitem(358,'W')	|								(358,'W')	|
delitem(354)	|								|	A
setitem(355,'X')	(355,'X')								|	|
setitem(351,'Y')	|								|	(351,'Y')

(7 marks)

Apply the hash function to the key, and then compress the result. So `setitem(354,'P')` *does not go into cell 4. In this example, the array is of length 10, so we compress by applying mode 10.*
354 hashes to 6529, which compresses to 9, so (354,'p') goes into cell 9..
Need to include both key and value – if key is not included, then you cant search for the item by key, and can't recognise a 'setitem' command that is meant to overwrite the existing entry for the key; if you don't include the value, you can't return the data.
There is at most one entry per key, so `setitem(358,'W')` *must replace the previous entry (358,'Q').*

Total: 30 marks

END OF PAPER