# Can someone restore the setting of the pages????

# 

# **COMP3506/7505: 2018 exam an**s**wers**

## [**UQAttic**](http://uqattic.net)

## **Get more out of your study time. Join UQAttic's revision chat.**

#### [**Other exam papers**](https://docs.google.com/folder/d/0B6_D4T6LJ-uwYzY1YWMzNjYtMzUyZC00OTEyLWJlMjktOGExYWUwOTc4NDE3/edit)

### Please **contribute** to these documents.

### If you're looking for an effective way to familiarise yourself with the course material, you can't go past collaborating with fellow students. We have laboured to put these up, and so at the very least point out where you think we are wrong!

### You'll get more out of the course, you'll do better in the exam, and other students will benefit from your input as well.

### To get editing permissions, simply go to the [chatroom](http://uqattic.net) and provide us with your Google Account address.

### **Style.**

### Type answers in blue beneath each question.

### If you're unsure of your answer, highlight your answer text then hit Ctrl+Alt+M to create a comment beside the text. Once you're satisfied with the answer, click the "Resolve" button on the comment.

### If you want some extra explanation from someone else on their answer, highlight the other person's answer and repeat the procedure above.]

### **Communicate.**

### Head over to [uqattic.net](http://uqattic.net/) and click "Chat Now!". You'll find a chatroom full of students just like you. Talk about a revision document (like this one) or swap prep tips. If you have your own IRC client, point it to irc.uqattic.net, port 6667, channel #attic.

# Question 1. Arrays and Linked Lists

Arrays store a set of values by index using O(*n*) space, where *n* is the largest index of the array. In some cases, most indices of the array are *null*, wasting space; such arrays are called **sparse arrays**. In other words, the number of non-default values stored *m*, given the array A that consists of, is much smaller than the largest index *n*.

[1, *null*, *null*, *null*, 2, *null*, *null*, *null*, *null*, *null*, *null*, *null*, *null*, *null*, *null*, 3]

we have *n* = 16, *m* = 3. We can instead store the data using a doubly linked list, storing each non-null value as an (index, value) pair. Since we only store the nonnull values, this list implementation gives us O(*m*) storage.

a. [4 marks] Draw a doubly linked-list representation of the data in *A*, showing both the structure and the data.

| I think the tail pointer should be pointing to the last element, not the other way around +3 |
| --- |
| Don’t the nodes also have to include their respective index? (Linked lists have no index) Revision states you need index?  It should store (0,1), (4,2), (15,3) storing (index, value) as requested + 1  You should store the index as in sparse arrays, the empty nodes are usually meaningful. -1 |
|  |

l

b. [2 marks] Explain one disadvantage of storing *A* using a doubly linked list instead of an array?

| Either of the following:   * **Slower lookups:** arrays can be accessed in O(1) time, whereas linked lists require O(n) time (O(m) in this case).**Memory overhead:** each linked list node stores pointers to the adjacent nodes in the list. |
| --- |
| Memory overhead compared to an array. Doubly linked lists store two pointers along with unit of memory. +1  Also bad access time. +1  I am wondering about LinkedList. If index is already given, like LinedList.get(10), this operation should be O(1) and thus the same as Array. (-1) -1 -1 -1 -1//  Assuming you know the position, for singly linked array, it is O(n), for doubly linked array, it costs O(1) time. (-1) -1-1-1-1-1  Depends how the list is implemented. The diagram above has no way to look directly at the middle element without first locating it by looking at the first or last elements. NB the LinkedList in Java 8 does a traversal from head or tail to the specified index for all index operations, which is O(n). |
|  |

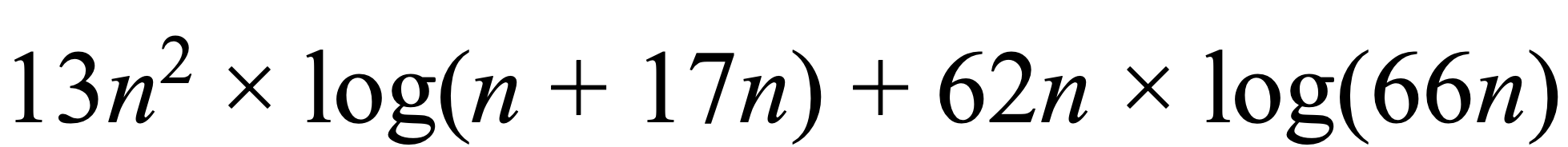
c. [4 marks] Explain why a linked list is usually implemented as a doubly linked list, rather than as a singly linked list.

| While both have lookup Big O(n)s, in practice being able to traverse a list backwards means that one doesn’t hav[e to restart searching from the beginning. Additionally, the overhead for the extra address is minor. +7 Also deletion becomes O(1) instead of O(n) (assuming your holding pointer to node)?? (+1)+1(because you are already in the right position(the node to be deleted), so you don’t have to search from the head to the previous node, so O(1)) No. When deleting nodes you save pointer to previous node during traversal, then that nodes next = next.next. Links in 2 directions mean you can optimise traversal - e.g. if you want 2nd last node, you can traverse backwards as opposed to forwards. Only tail deletion is in constant time. I think the reason is that in linked list, when you delete a node, you need to update the previous node’s ‘next’ pointer to the deleted. Please correct me if I’m wrong. +1  Complexity of insertion and deletion at a given position is O(n / 2) = O(n) because traversal can be made from start or from the end (Geeks for Geeks). When you’re deleting an element, you don’t have a reference to its node object, only the value it holds. Hence, in a doubly-linked list, you still have to do a traversal to find the corresponding node object for the value, making it O(n). |
| --- |

d. [5 marks] Sparse arrays can also be implemented efficiently with hash tables. Describe how a hash table might be used to store a sparse array with O(*m*) storage.

| Hash the index of the element as if it were in an array, then load it into the Hash Table using the index as the key, and the element as the value. Need to double hash table from size O(n) to size O(m) (ie: by setting table size = 2m). As the number of loaded elements grows, so will the size of the table *linearly with m*. Important to note here that growth should be incremental, not doubling (thus O(m)). w  ^ I don’t think that matters… even if it is doubled, the storage is 2m = O(m). Can someone confirm? +1  Think it’s important to mention using probing, since LinkedList implementation isn’t O(m) space?  1. Use index of the simulated array to as hash code +2  2. Add an element to the sparse array by hashing them (and getting them)  3. If the hashmap becomes too full or empty, resize. Do this as a normal hashmap would (dynamically).  4. When iterating in order, iterate over the key range of simulated indexes)  5. Bonus mark for insightful comment/ picture/ example/ being clear/ using big-O access? |
| --- |
|  |

# Question 2. Asymptotics

Given a function .

a. [4 marks] Show your working to determine the big-O notation of the function.

| Big O - n^2 log(n)  ,  +1 +1, provided we aren’t supposed to do formal working for this +1 |
| --- |
|  |
| This is the mathematical method. When c is greater than 13 (the larger the better), Derivative of the right side will be greater than the left side and there will be a n value make big O exist. Also when c is less than 13, the derivative of the right side will be negative and the output of the right side will be less than 0 (output of the left side will be positive). Which means big omega exist. In conclusion, big theta exist. (this one is a little bit complicated ) |
|  |

b. [4 marks] Show your working to determine the big-Ω (omega) notation of the function.

| Big-Ω +1    Omega() is correct as well! 、、、  for all n >= 1, c = 1 Ω( log(n)) is the correct answer because we need the *tightest* lower bound. +1 |
| --- |

-

c. [4 marks] Can big-Θ (theta) be determined for the function? Explain why or why not.

| Big O and big Ω bounds are the same for this function, therefore theta exists and will be . +1 |
| --- |

d) Explain the relationships between big-O, big-Ω and big-Θ notations. Also, explain how they can be used in evaluating the suitability of an algorithm for a particular context.

| Big O describes a theoretical upper bound on an algorithm. Big Omega describes a theoretical lower bound on an algorithm.  Big Theta => when O and Omega are equal. Tight bound.  Important, as practical experimental runtime can differ depending on hardware used! Thus, best to use a theoretical approach to compare algorithms. +1  Asymptotic analysis also does not require an algorithm to be implemented, saving time spent on development for testing.  Big O is useful for determining the suitability for an algorithm in time- or memory-critical contexts. An algorithm that has good expected time but poor worst-case time may be a poor choice for a real-time system, where there is a hard upper bound on the amount of time/space available. |
| --- |

# Question 3. Binary Trees

The following algorithm takes a non-empty proper binary tree (tree) and returns a List of all of the external nodes (leaves) in the tree. Recall that each node in a proper binary tree is either external, or has exactly two children.

| Algorithm getExternalNodes(tree):  return getExternalHelper(tree, tree.root())  Algorithm getExternalHelper(tree, node) o  externalNodes ← new empty Linked List  if tree.isExternal(node) then  externalNodes.add(node.getElement())  else  leftLeaves ← getExternalHelper(tree, tree.leftChild(node))  rightLeaves ← getExternalHelper(tree, tree.rightChild(node))  for leaf in leftLeaves do  externalNodes.add(leaf)  for leaf in rightLeaves do  externalNodes.add(leaf)    return externalNodes |
| --- |

The method add, from the List ADT, appends the given element onto the end of the List, so that for a Linked List implementation, you may assume that method add has O(1) time complexity.

a. [1 mark] How many external nodes can there be in a non-empty proper binary tree containing *n* nodes in total?

| **Note: External nodes are also called leaves.**  Draw out some proper binary trees. Try n=1, 2, 3, 4, 5, 6, 7, 11. You will find that you cannot have valid proper binary trees on some of those n. Try to relate the number of leaf nodes to n. You should get .  Confirmation: [slide 2 of this PDF](http://courses.cs.vt.edu/~cs3114/Fall09/wmcquain/Notes/T03a.BinaryTreeTheorems.pdf), point (d). If you have nulls on non valued leaves is it not all the same anyhow. sup |
| --- |
| I think it should be (n+1)/2, like,3 nodes tree has 2 . +8  (round down???) no need to round down (floor) as it is a proper binary tree (n will always be odd)  (n+1)/2 Lecture slide week 4 page 70 |

b. [6 marks] What is the worst-case time complexity of the above algorithm getExternalNodes, given that there are *n* nodes in the parameter tree? Express your answer in terms of *n* in big-O notation, making the bound as tight as possible and simplifying your answer as much as possible.

| Points to  note:   * The tree is a **proper** binary tree, meaning nodes have either zero or two children. |
| --- |
| O(n2) in the worst case where all internal nodes (except the root node) are left-children (or right-children) of their parents.  Worst-case for the tree with n=7 nodes:  ~~WRONG~~ (Why? It is proper binary tree) +1 +1  Would only be wrong if we wanted a complete BT +5  O(nlog(n)) in the best case where the tree is a complete binary tree (or any tree with a logarithmic height). +5  So then what’s the answer? +5  (n+1)/2 + (n+1)/2 = (n+1)/2 + (n+  (n+1)/2 + 2 + 3 + …. + (n+1)/2 = sum(2 to (1)(n+3)/8  Thereby in the worst case this is a n^2 function. +henry, I got this as well but not certain if correct  You guys are missing the point of the n^2 complexity -> it’s the internal loops, you have to find a tree which will exploit this vulnerability. +1, think about it as height = n recursive calls as it is not a complete tree, multiplied by n as each call iterates through the entire subtree below it.  <- TI don’t think so -> once you’ve added the external nodes, it doesn’t grow anymore. You just add the same list at each step  The runtime variables are the cost of traversing a tree that is a chain of length (n-1)/2 [The worst case number of levels in a tree] multiplied by the cost of adding each leaf to the levels linked list at each step, which is (n+1)/2 [The worst case number of leaves in a sub tree]  `  = O( (n-1)/2 \* (n+1)/2)  = O( n\*n )  = O(n^2)  Why is the right child always external. If youre on a multilevel graph wouldnt it also have T(n/2) both left and right sides? |

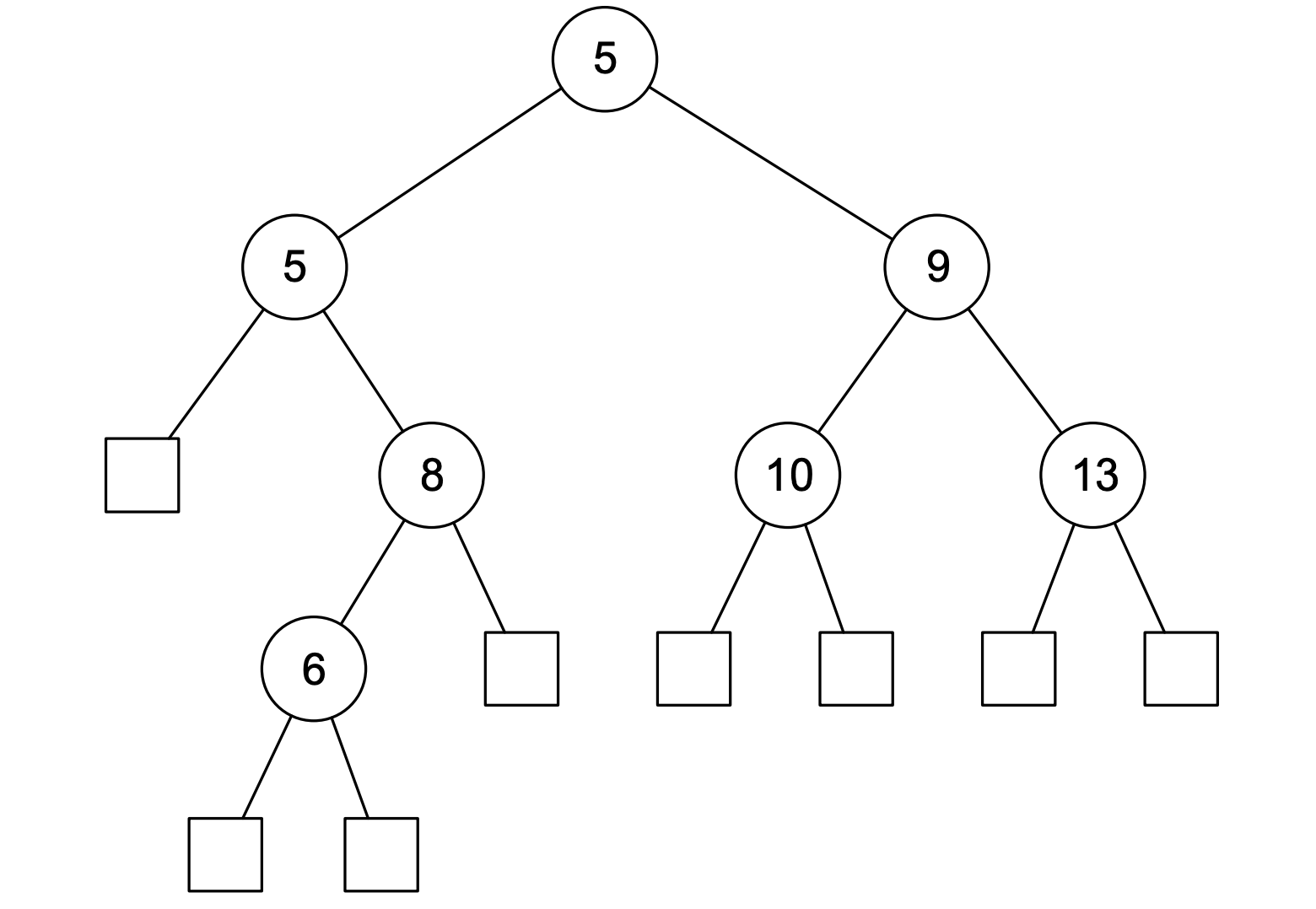
c. [5 marks] Can the algorithm getExternalNodes, be rewritten so that it is more run-time efficient? If so, provide a rewritten algorithm. Provide a justified explanation of its big-O time complexity. If the algorithm cannot be rewritten to be more runtime efficient, explain what are the limitations of the binary tree data structure that mean the algorithm cannot be any more efficient. If you believe that the algorithm could be rewritten to be more efficient, but cannot think of how to write it, provide an explanation of what aspects of the getExternalNodes algorithm you think could be improved. Given an explanation as to what type of time complexity you think could be achieved in a rewritten algorithm.

| **Proposed Answers** | **Commentary** |
| --- | --- |
| Options include:   1. **Modify the DFS to create a single linked list before the first recursive call.** Pass this list to the recursive call as a parameter. Each time the base case is reached (leaf node), simply append to this list in  time. As each node gets visited once in this post-order traversal, we get a running time of 2. **Perform a level order traversal (BFS).** Add additional logic during each visit of a node. If a visited node has no children (which can be checked in constant time) then add it to the output list. Add its children to the queue as normal, done in O(1) time. The runtime is the same as #1 and hence O(n) for *n* nodes. | +6 |
| Can’t. The reason is that this method will return a list of nodes, its time complexity has to be O(n), also, for n nodes tree, there will be (n+1)/2 external nodes to be returned. | +1 |
| How about in order traversal? We traverse all the node which is only O(n), e a lot of time is spent superfluously adding nodes to temporary lists in the original function. | +5  Could someone write the actual algorithm in pseudo code for this? |
| Perform a depth first search on the tree (think about it - a tree is just another graph), and add to the list if it is an external node! Thus O(n + m) time, where n is number of nodes, m is number of edges.  ^ Actually this can be simplified to O(n) since for any tree m = n-1. | +1 |
| Height of a tree is its lower bound. The height of the tree is O(Log N) and we must return a list of N elements. Therefore, this is asymptotically efficient. ??? No, We are not branching to one child at each node, we go to two. Can’t be log |  |
| From the tree construct a heap bottom up O(n). Get its height O(logn). Using height calculate the index of the first node in the exterior layer O(1) then iterate until the first null value adding nodes to our list O(n). This method is O(n). | Mid |
| Using breadth-first traversal for O(n) time Algorithm getExternalNodes(tree):  Q ← new empty queue  L ← new empty linked list  Q.enqueue(tree.root())  while not Q.isEmpty():  node ← Q.dequeue()  if tree.isExternal(node):  L.add(node)  else:  Q.enqueue(tree.leftChild(node))  Q.enqueue(tree.rightChild(node))  Return L |  |
| Convert the tree to an array using post traversal without worrying about empty nodes. O(n) operation as we need to traverse through n elements.  Then iterate through the array and stopping when (n+1)/2 external nodes are found. O(n) operation as external nodes can be at the end of the array.  Therefore, O(n) + O(n) = O(n) time complexity.6 |  |
| Can’t we just declare the externalNodes list in the getExternalNodes function, pass that into the Helper function, and delete the “for leaf in leftLeaves” and “for leaf in rightLeaves” parts?  Algorithm getExternalNodes(tree):  externalNodes ← new empty Linked List  return getExternalHelper(tree, tree.root(), externalNodes)  Algorithm getExternalHelper(tree, node, externalNodes):  if tree.isExternal(node) then  externalNodes.add(node.getElement())  else  getExternalHelper(tree, tree.leftChild(node), externalNodes)  getExternalHelper(tree, tree.rightChild(node), externalNodes)  return externalNodes |  |

### 

# Question 4. Heaps

Answer the following questions related to the following binary tree diagram.



a. [3 marks] Does the tree satisfy the ‘heap-order’ property? Why or why not?

| No. For a min (max) heap, the heap property requires that each nodes children be greater (less than) itself. Here, there is a violation where 6 is less than 8. |
| --- |
| No 6 < 8, which means 8 should be child of 6 6 is not greater than 8, i think this is the wrong way around.  Heap-order property: children >= parent. Therefore No, does not satisfy because 6 < 8. +7 |

b. [3 marks] Does the tree satisfy the height-balance property? Why or why not?

| No. The height of node 5's left and right trees are 0 and 2 respectively. This differs by more than 1, violating the height-balance property. +1  (You can consider the left band right tree heights as 1 and 3, depending on whether you include those square markers in the height. It's a matter of zero vs one-indexing, in a way.)  I believe we do include the empty nodes in height calculations if they are depicted in the image. The difference between branch heights for node 5 (not root) is still greater than 1, hence the heap-balance property is violated. |
| --- |
| No. Explanation below  No or Yes(confused about this). The property requires that the difference of the height of two children nodes which under the same parent should be no more than 1.  (Disagree)The difference between branch heights for node 5 (not root) is still greater than 1: the left child has a height of 0, and the right child a height of 2; hence the heap-balance property is violated.  I thought the answer was yes, tree satisfies the height balance property. Why? Because for any node, the right and left subtree heights do not differ by more than 1. -1 -1  I.e. (5) on the left has a height of 3, and (9) on the right has height of 2 (3-2=1). Therefore it’s all good  **If** the height of the left (5) was 4 then the height difference would be more than 1 (4-2=2) and therefore not satisfy the height balance property.+2 +1  Height-Balance property: for ***every*** internal node v, the heights of the children of v differ by at most 1.  Think about the tree of node (5). The lhs height is 0, the rhs height is 2. Differ by 2. +1+1+1  Q: (what I confused is left child of 5 is null, should we count that height? )  A: yes +1 |
| **My take- No it does not. Look at slide 87 for Week 6. She goes through a similar tree.** |
| heights drawn in red. |
| Here we see that the null nodes have height of 0. +1 |
| So no.  I agree with this one now, the answer is no. +17+1 |

c. [4 marks] Draw the array-based representation (as described in lectures) of the above tree.

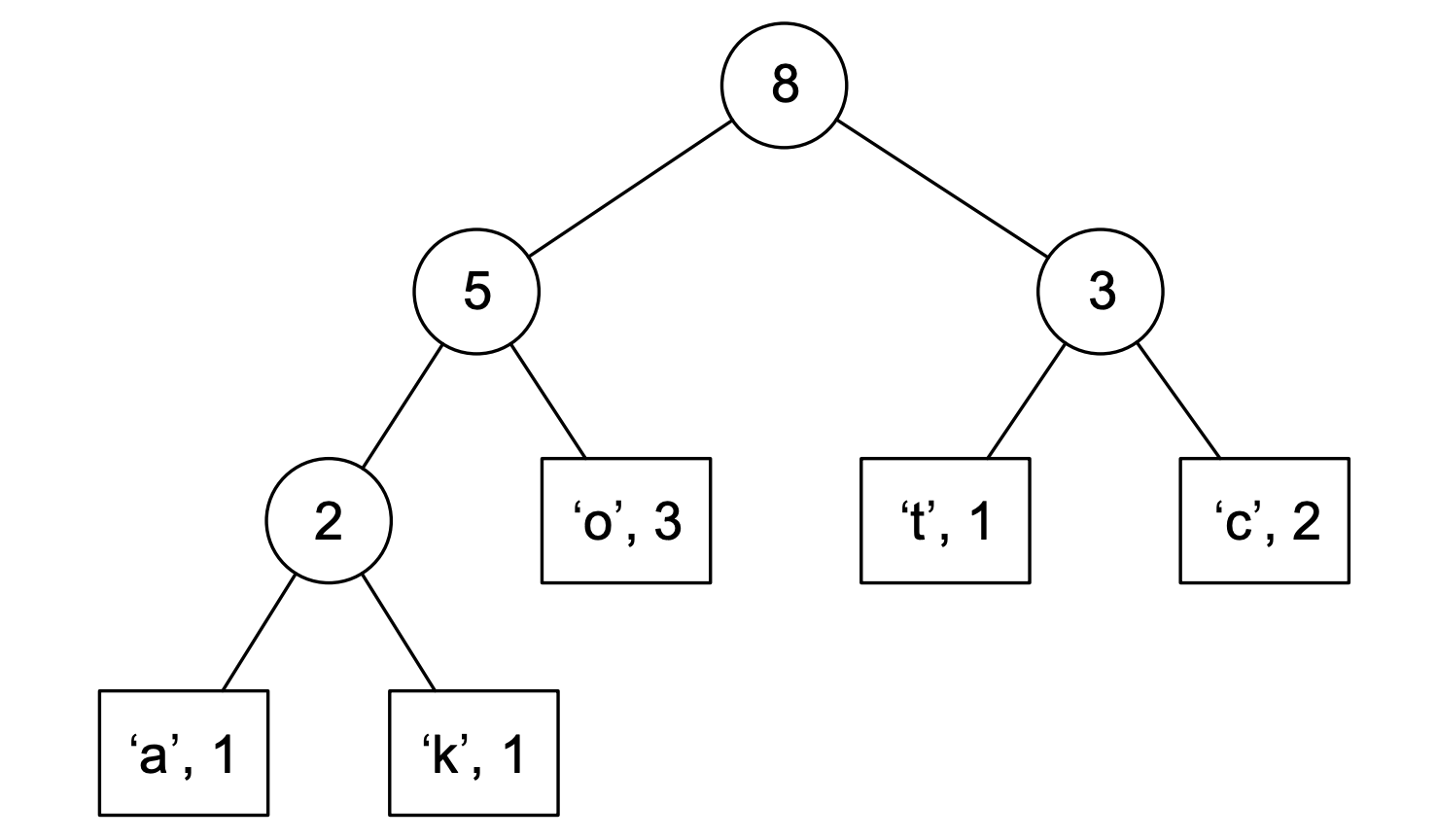
| **Proposed Answers** | **Commentary** |
| --- | --- |
| (5) → (5) → (9) → (NULL) → (8) → (10) → (13) → NULL → NULL → (6) → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → NULL → to infinity and → beyond | I think the null can be ignored -1(No in the tutes Henry said to include the nulls +1 +1)  \*  43  2 |
| I’m pretty sure both ways are correct as question just asks for “array-based”   | 5 | 5 | 9 | 8 | 10 | 13 | 6 | | --- | --- | --- | --- | --- | --- | --- | | +1 -7-1A |
| Need to consider that arrays can be sparse and can include nulls (as listed in q1). Should clarify in revision session | +1+1 |
| [5, 5, 9, null, 8, 10, 13, null, null, 6, null, null, null, null, null]  Refer to week13 tutorial q7(a). What about the two external nodes under 6? At that point, that entire row in the tree is null. No value in doubling the array size just fill it with nulls (they’re only placeholders anyway). | +21 |
| What about 6’s children? I would argue (just to be EXTRA safe :P)  [5, 5, 9, null, 8, 10, 13, null, null, 6, null, null, null, null, null, null, null, null, null, null, null]  This includes children of 6 @ index 19 and 20 as technically they are “present” in the tree.. | +3+1  (See week 4 slide 106) |
| The above are all wrong, because y’all did not map the parent-child relationship.  2^(H+1) - 1 is the size of the array, u dont need that much nulls.  In this case 2^(3+1) - 1 = 15, the size of the array should be 15, i only drew 11 cuz im lazy af and 11 is sufficient to map the child-parent relation at this point.  Is height not equal to 5? (Note height does not start at 0) Granted you consider the empty nodes hence an array of 63 would be needed at a **maximum**, however in our case a lot of these nodes are not listed or are empty, taking for instance that there are no further nodes to the left of 5 (bar the null). If we look at the week 4 lecture it just indicates the index of each node location however nulls would still be included for the locations of the represented index, the answer in purple above does it best. | +1 (See week 4 lecture slide p78, 2022)  -1-1 |
| If we are including null sentinels in the array representation (which I would assume we do, as they are visible in the image) we assign each node an index based on the rank of the parent, and whether the node is a left or a right child. Hence the array representation should be:  [5,5,9,Empty, 8, 10, 13, Null, Null, 6, Empty, Empty, Empty, Empty, Empty, Null, Null, Null, Null,Empty, Empty]  Where “Empty” denotes an empty node (as displayed in the image), and “Null” denotes an index that does not contain anything (Similar to 7, and 8 in the example image above) |  |

d. [2 marks] List the nodes in the order that a post-order tree traversal would visit them.

| Source: [[link](https://docs.google.com/presentation/d/1QJ2H8WKwNn9FXYTM38sZEp8vI_AEkHymQNYlDfohe_M/edit#slide=id.g54bbc17829_0_203)] |
| --- |
| Using the 'useful visual trick', circle the entire tree and then see when we cross the right of each node. Thus we visit in the order of: 6 8 5 10 13 9 5. |
| 6 8 5 10 13 9 5 +25  Q: Can anyone tell me why not include squares?  A: Squares are leaf nodes which don’t store anything in a binary tree.  Q2: To clarify, we don't include them because they are null, or because they are leaf nodes (in a binary tree)? Because they are null. Visiting a null node isn’t very interesting. You DO visit binary tree leaf (external) nodes if they are not trivial (not null, or a similar null sentinel).  Yes, if this was an array implementation then we would, but not for just listing traversal. They’re simply placeholders for our “proper” binary tree structure (i.e., always 2 children unless external). |

# Question 5. Huffman Trees

Below is a Huffman tree generated from the frequencies of the characters in the string “cockatoo”.



a. [4 marks] Using the Huffman tree above, determine the binary code required to transmit the text “cockatoo”.

| Answer should be: 11 01 11 001 000 10 01 01 +29  This slide is useful: (0 represent moving left, 1 represents moving right down the tree) |
| --- |

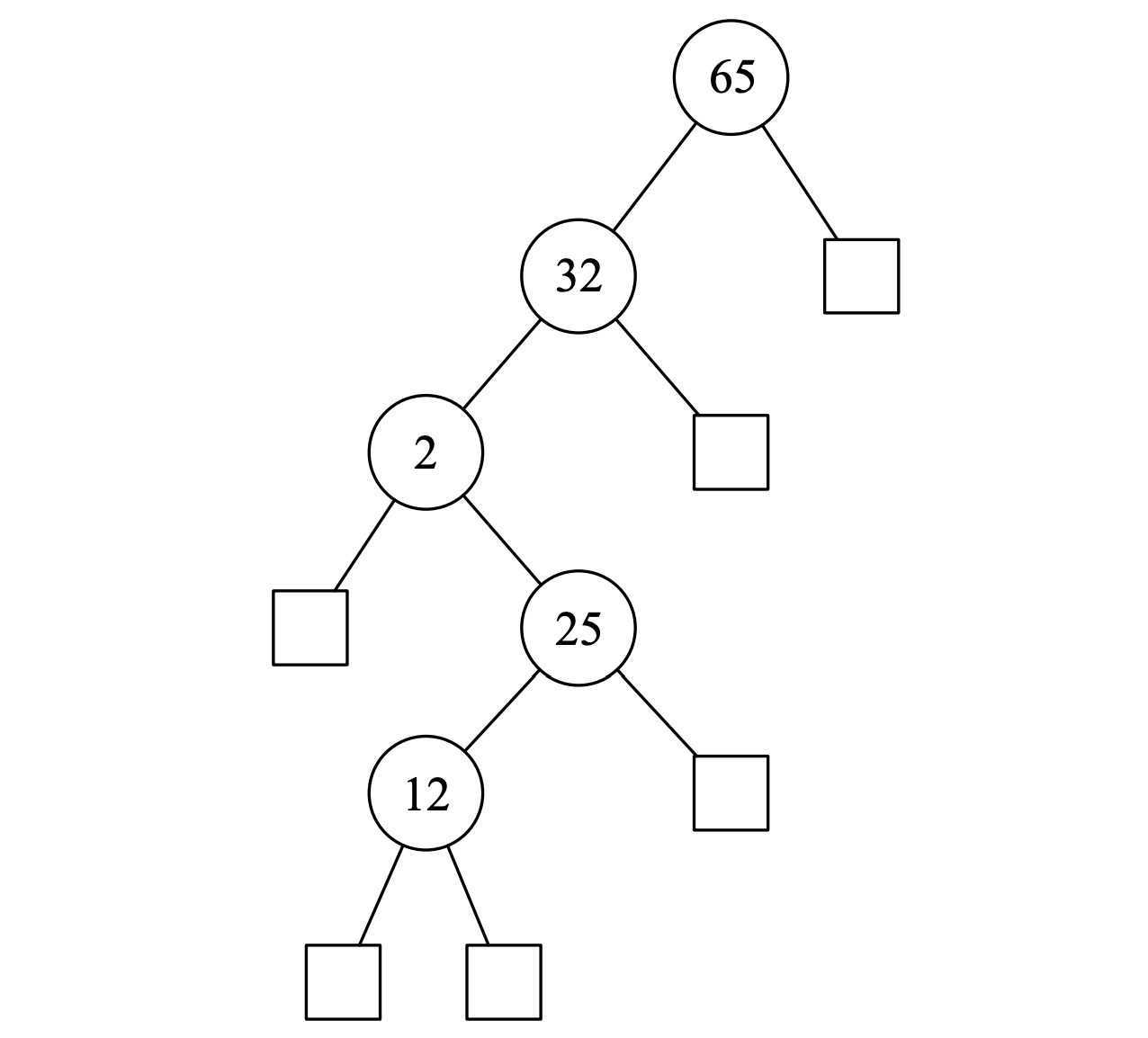
b. [6 marks] Using Huffman’s algorithm, construct a Huffman tree that defines an optimal prefix code for the string “woolloomooloo”. Show the intermediate states of the priority queue.

++

| (This is just the finished state, soz. Do the intermediate steps yourself :) ) +1 :^ +4 )  Should root node be 13 instead of 8?+2+132+1 |
| --- |
| Is it still right if i had that exact thing but mirrored?? Or do we have to build it right to left like this?  Should be fine going the other way - in fact in the lectures the “lesser” value was always on the left (think BST) - i would go that way just to be safe. +5+1  This is something to ask on monday^ The higher value should be on the right. +1+1  What if the values Gwere equal (swapping m and w in the above tree)? Doesn’t matter if they’re the same. It’s more significant for non-equal values that the larger values are on the right.  Shouldn’t 8 be 13 since 8+5? +5 -10  I agree, this is the usual way to do this, unless lecture slides show any different, root node should be 13 instead of 8.+1  Lowest value goes to the left, higher to right (at each stage). In this example, if o = 4, then o would be on the left in the final tree( because 4 < 5), and the tree with l,w,m would be on the right +1 |

# Question 6. Binary Search Trees

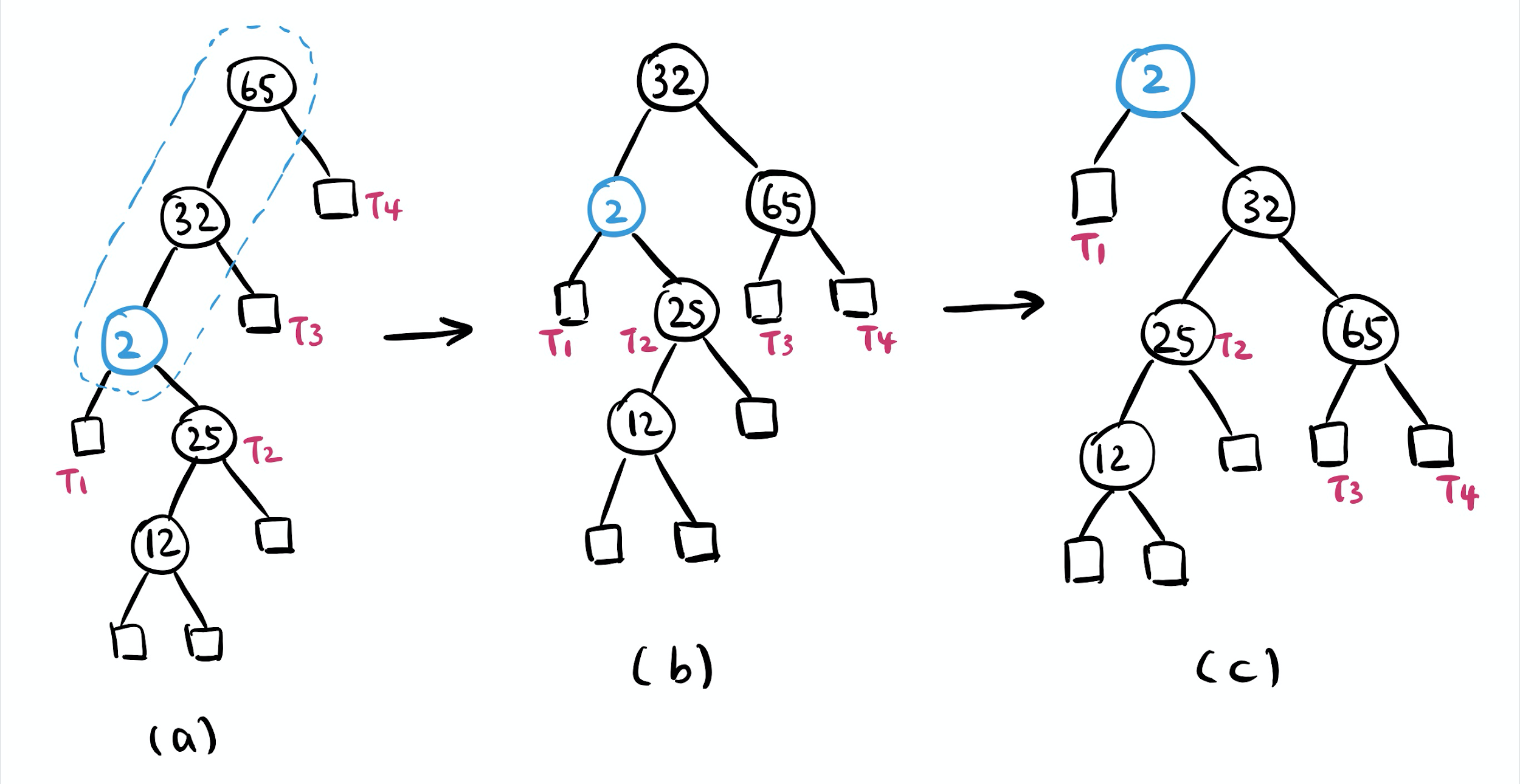
a. [3 marks] Is the following tree a valid representation of a splay tree after inserting the keys 12, 25, 2, 32, 65? Why or why not?



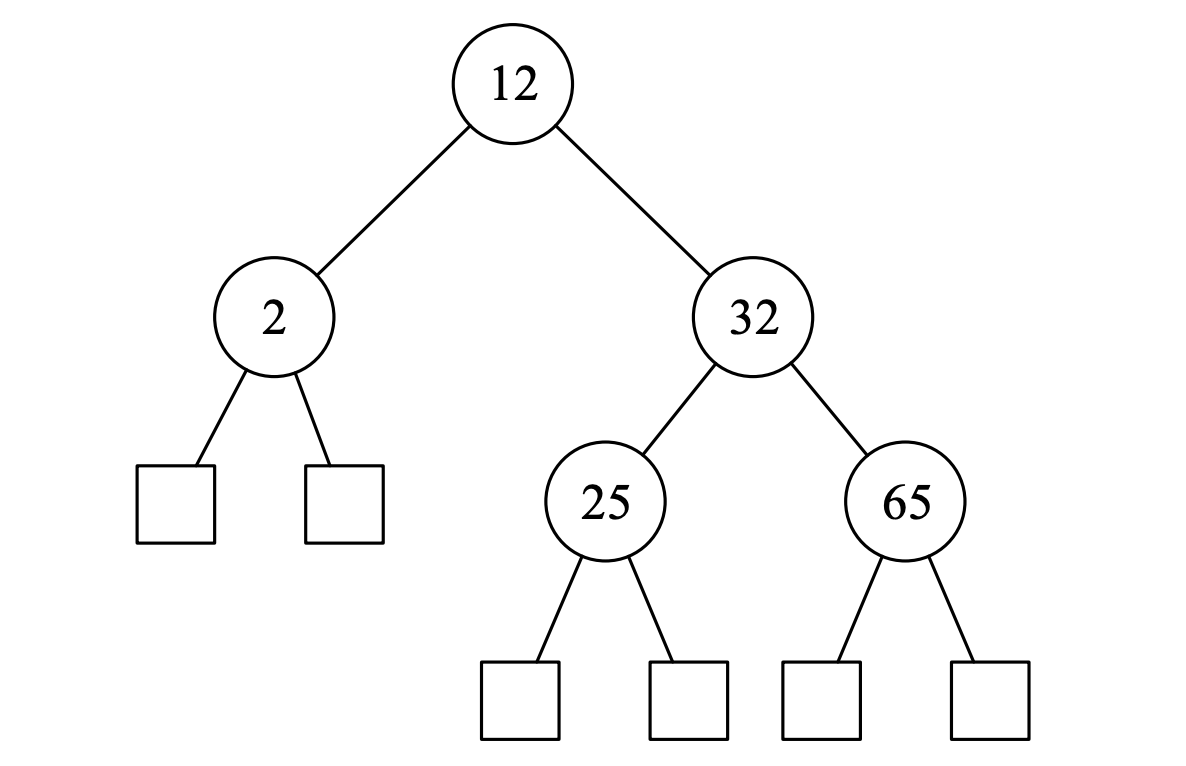
| Yes try it on this [web](https://www.cs.usfca.edu/~galles/visualization/SplayTree.html) why though? A splay tree moves the most recently accessed values to the root of the tree  (Can use other websites to visualise algorithms <https://visualgo.net/en>. Very useful)  Why does the example in the lecture slides (slide 19 w7) give a different shaped tree than to the one on the above website? \*/- Check it again, I think it’s same.+1  j  Yes since it is a valid Binary Search Tree, splay trees are unbalanced BST’s which this tree is so yes it is a valid splay tree. -1  You guys are missing the point of the question. It is not asking specifically if this is a valid splay tree, it’s asking if the above tree, a valid representation of the splay tree after inserting those values.  So the correct way to check is to construct the splay tree yourself, inserting and splaying nodes in the provided order, and then just observing if you get the same tree (which you will see the answer is yes).+1+1 -1+1+1  There is no need to construct the splay tree yourself.  The splay tree is valid as long as:  1. the root is correct;  2. It’s an BST. （-10086）  (See this ed post: <https://edstem.org/au/courses/9281/discussion/1130552>)  It’s a little stronger than that: the order from root needs to correspond to the order of access in the nodes (since that’s the order they’ll be splayed in). Since they do in fact run in the reverse order of insertion, it’s a valid splay tree.+1 |
| --- |

b. [6 marks] Draw the splay tree after a search for the node with the key 2 has been performed on the tree above. Describe what steps were involved in making the change to the tree’s structure.

| 1. Key 2 is a left-left grandchild of the root, so it is a zig-zig situation. 2. Perform a right-rotation about key 65 (the grandparent of key 2). 3. Perform a right-rotation about key 32 (the parent of key 2). 4. Since key 2 is already the root, we stop the splaying process.   For (a) isn’t actually a zag-zag? on the lec slide with the flowchart there were typos and left-left and right-right both said zig-zig, and right-left and left-right both said zig-zag. Idk correct me if i’m wrong but that flowchart slide is wrong right?? +2+1  There’s no difference between “zig-zig” and “zag-zag” and all they do is just to differentiate “zig-zag” which has a bend and a simple “zig”. If you see on textbook p.488 it also regards both left-left and right-right as “zig-zig”, and both left-right and right-left as “zig-zag” so I believe it shouldn’t be bothered too much.  In the context of this course, a double right rotation is called a ‘zag zag’ and a double left rotation ‘zig zig’ hence I would argue that the distinction between ‘zig zig’ and ‘zag zag’ is significant.  Additionally, I noticed that the splay [tree visualization guide](https://www.cs.usfca.edu/~galles/visualization/SplayTree.html) describes ‘zig zig right’ and ‘zig zig left’ operations, whereas for this course, direction seems to be implied by the use of ‘zig’ and ‘zag’ for left and right rotations respectively.  <https://www.cs.usfca.edu/~galles/visualization/SplayTree.html> ← helpful for visualising and tells you operations, just saying. |
| --- |

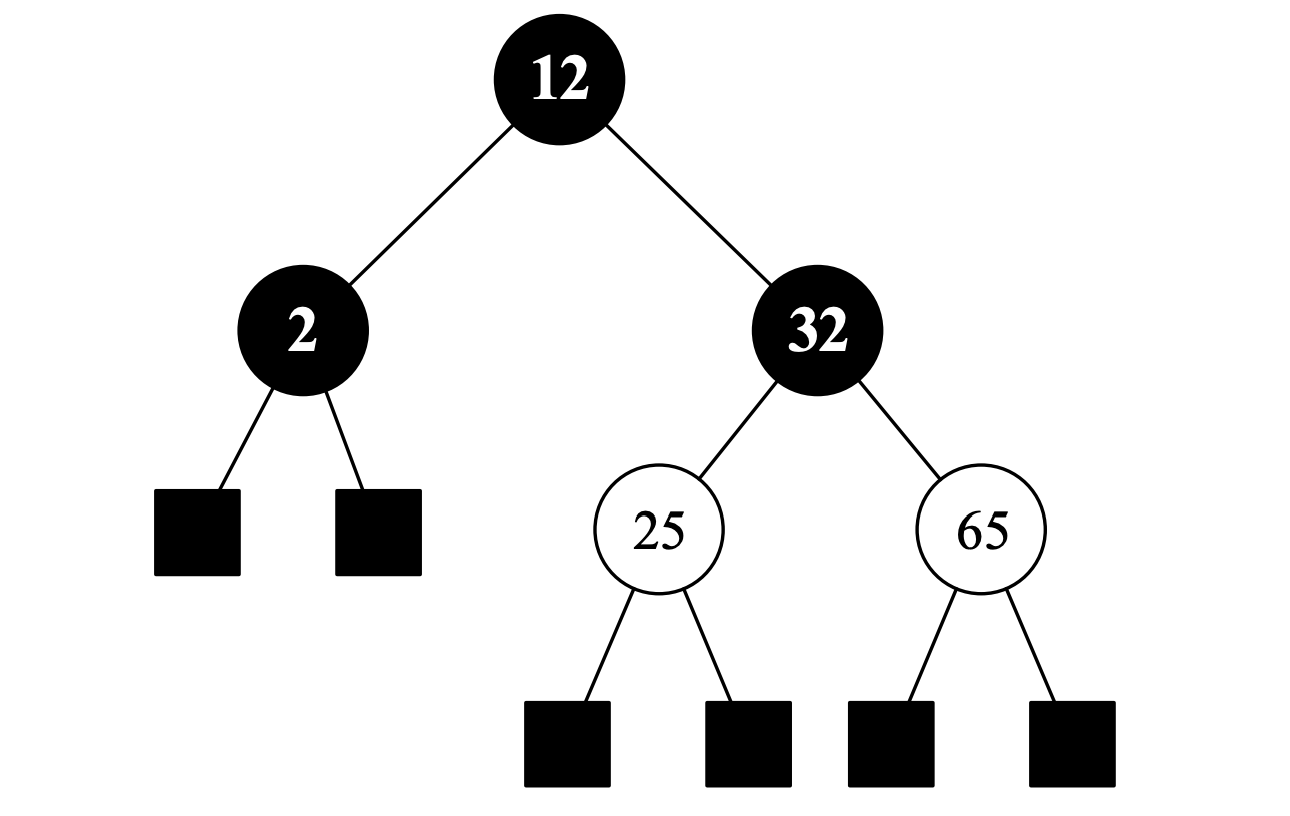


c. [6 marks] The following tree is an AVL tree after inserting the keys 12, 25, 2, 32, 65. Draw the resulting AVL tree after the node with the key 2 has been deleted. Describe what steps were involved in making the change to the tree’s structure.



| 1. Key 12 is the first unbalanced node encountered while travelling up the tree from the deleted spot. Key 32 is the child of the root with greater height; key 65 is the child of key 32 with greater height and also on the same side as key 32. 2. Perform a trinode restructuring (single rotation) of key 12, 32 and 65 to restore the height-balance property.+1   Siccccck mate lookin’ good there  Not sure how this can be the graph considering the imbalance is from node at left subtree of 12 and 25. This is what I got.    Would both graphs be valid here?  (I believe both are valid as your result still maintains AVL tree’s properties.)  No, this is not valid, 25 is not the one that really causes imbalance, you can’t rotate it, it is not AVL’s algorithm. You need to find out the one causing the imbalance and operate on it, which is 12 in this case. |
| --- |

d. [6 marks] The following tree is a red-black tree after inserting the keys 12, 25, 2, 32, 65. (Black nodes are filled in black, red nodes are filled in white.) Draw the resulting red-black tree after inserting the key 15. Describe what steps were involved in making the change to the tree’s structure.



| 1. After inserting key 15, a double-red problem occurred. 2. Since the sibling of key 25 is a red node, we perform a recolouring to remedy the problem.   +11  I think it’s wrong, because its not balanced???  Tree has the same “black length” along each path, and is balanced.  Both solutions (above and below) are valid red-black trees.  This can be checked again by converting each to a 2-4 tree, and seeing that each is valid.  25  12 32  2 15 65 ←-? Correct? No, original answer is right  I think the above is correct too. In the original answer op ‘pushed up’ 32 into the top node. But it is equally valid to ‘push up’ 25 into the top node.  No it isn’t. 15’s uncle is red, so you have to do recolour. The original solution is the only correct answer  Uncle related through marriage or blood? |
| --- |
|  |

# Question 7. Sorting

a. [6 marks] The lectures described a merge sort algorithm, which is provided below. Describe what would need to occur at each step of the algorithm to perform a merge sort of data in a file in external memory. Assume that the entire contents of the file would not fit into the computer’s memory.

| Algorithm mergeSort(S):  Input sequence S with n elements  Output sequence S sorted in ascending order  if S.size() > 1  (S1, S2) ← partition(S, n÷2)  mergeSort(S1)  mergeSort(S2)  S ← merge(S1, S2) |
| --- |
| Algorithm merge(A, B):  Input sequences A and B with n÷2 elements each  Output sorted sequence S of A ⋃ B  S ← empty sequence  while ㄱA.isEmpty() ∧ ㄱB.isEmpty()  if A.first().element() ≤ B.first().element()  S.addLast(A.remove(A.first()))  else  S.addLast(B.remove(B.first()))  while ㄱA.isEmpty()  S.addLast(A.remove(A.first()))  while ㄱB.isEmpty()  S.addLast(B.remove(B.first()))  return S |

| The idea is you process the file as chunks, only keeping small chunks in memory at the same time.  In terms of the algorithm merge-sort involves partition and saving the halves as separate files.  Merge would take in files and read from each saving (streaming) to the result file to not keep things in memory.  ‘External sorting typically uses a hybrid sort-merge strategy. In the sorting phase, chunks of data small enough to fit in main memory are read, sorted, and written out to a temporary file. In the merge phase, the sorted sub-files are combined into a single larger file.’    Could you use yield or something similar to do this as well?  <https://www.geeksforgeeks.org/external-sorting/>  <https://en.wikipedia.org/wiki/External_sorting#External_merge_sort>  Probably an A-level question.  A-level? A-level meaning of a very high standard. Particularly regarding the traditional 5-letter grading system often used in schools (with A being the highest level). |
| --- |

b. [4 marks] In the lectures it was stated that the worst case time complexity of merge sort was O(*n*∙log *n*) and that the time complexity of quick sort was expected to be O(*n*∙log *n*). It was also stated that, in most cases, quick sort would execute a little faster than an in-memory merge sort. Explain why this is the case. What characteristics of the quick sort algorithm are responsible for this speed improvement?

| Quicksort is in like virtual memory environment (stack) , merge sort sequential data access in memory less efficient ? merge sort uses extra space for each temporary array when the data is partitioned into sub arrays and recursive divide and conquer calls, quicksort doesn't because it is an in place sorting algorithm. Hence no additional memory is required for this in place sorting algorithm?  The reason why quick sort is faster than merge sort in many cases is not because of reduced overhead but because of how **quicksort accesses data, which is a lot more cache friendly than a standard mergesort**.(not sure whether it is correct) |
| --- |
| H xol up QS isn’t in place all the time tho? (Quick sort is an i) |
| Another take - What’s one difference between quicksort and merge sort? In merge sort, no matter what, you split in half and work all the way down to the bottom, then merging back up. In QS however, we split based on that pivot, and instead of sorting ALL, we sort on two halves MINUS that pivot! Thus, on very large data sets, there is one less number to work on every level of the algorithm compared to merge sort, thus it can execute a little faster. (Confirmed by Henry <3 <3 <3) +4 <3 +10+1 |

c. [2 marks] Explain why the quick select algorithm is expected to have time complexity of O(*n*). Why is the expected time complexity of quick select O(*n*), while the expected time complexity of quick sort is O(*n*∙log *n*)?

| (A mathematical proof from the lecture, probably overkill this question)  Probabilistic Fact #1:  Expected number of coin tosses required in order to get one head is two.  Probabilistic Fact #2: Expectation is a linear function.  E(X + Y) = E(X) + E(Y)  E(cX) = cE(X)  Let T(n) denote the expected running time of quick select.  By Fact #2,  By Fact #1,  That is, T(n) is a geometric series    So T(n) is O(n). |
| --- |
| [From Wikipedia] ‘*Quickselect uses the same overall approach as quicksort, choosing one element as a pivot and partitioning the data in two based on the pivot, accordingly as less than or greater than the pivot. However, instead of recursing into both sides, as in quicksort, quickselect only recurses into one side – the side with the element it is searching for. This reduces the average complexity from O(n log n) to O(n), with a worst case of O(n2).*’  This means that Quickselect still does log(n) steps, but at each step it (on average) only has to check half of the previous step. [So n + 1/2n + 1/4n + 1/8n …< 2n (in total/sum)].  This means we get O( log(n) + 2n)  = O(n)  Whereas Quicksort must still check the whole array at each step so that O(nlog(n)) |
| Did 2022 content cover quick select? No, it’s in week 12, which is not examable. Could not be happier to hear that+1 |

# END OF EXAM

# YOU ARE ALLOWED DOUBLE SIDED A4 NOTE NO FONT SIZE LIMITED VERY COOL!

Not for 2021 tho ;( +1 +1 +1 biggest of sads +sad :( +sadsad big sad

SADDDDD

Cryyyyy

No pseudo code in 2018 right?