**Name: Patrick O’Brien**

For each of the below questions, write a short sentence or two to express (in your own words) your answer. Keep the answers short, but use complete, correct, English sentences.

If it helps to clarify the questions, feel free to mentally prefix all the questions with the phrase "According to the video…"

1. After you’ve watched all the videos, please answer this question:  
   Of all the videos that you watched, if you could pick one video to be re-recorded by the instructor outside of class which would you choose? Why?  
   (Keep in mind the recording outside of class will omit any pauses from the instructor answering student questions, have less hemming and hawing, etc, and generally be more concise)

|  |
| --- |
| < Write your answer here > |

**VIDEO: BST - Remove (Concepts)**

1. What is the major topic for this video?

|  |
| --- |
| Removing values from a binary search tree |

1. How many cases do we have to worry about? What are they (in a nutshell)

|  |
| --- |
| 1. 0 child 2. 1 child 3. 2 children   The subtrees of the root value |

1. When asked to remove the value ‘15’ from the tree in the video, what is the first step?

|  |
| --- |
| The first step is to identify where the value is in the tree. |

1. When walking down the tree (looking for the target node that contains 15) how many pointers/references do we maintain? What are their names?

|  |
| --- |
| Par and cur, 2 references. |

1. Which case does removing the value ‘15’ from the tree in the video illustrate?

|  |
| --- |
| Case A – having 0 children (removing from the end) |

1. Case 1 is similar to removing what from a linked list?

|  |
| --- |
| Case 1 is similar to removing a node from the end of a linked list. |

1. Which case does removing the value ‘10’ from the tree in the video illustrate?

|  |
| --- |
| Remove 10 from the tree in the video demonstrates case B, where there is 1 child. |

1. When removing 10 from the tree, how do we remove it? Please explain this both intuitively, and by talking about the left/right references in the parent.

|  |
| --- |
| You go up to the parent node, and splice out the node. If parent.left is the same as cur, take that value and replace the connection to the child of 10. |

1. Why is it safe to remove 10 from the tree?

|  |
| --- |
| It is safe to remove 10 from the tree because, everything less than 10 is less than 10 and also less than 20. |

1. Which case does removing the value ‘50’ from the tree in the video illustrate?

|  |
| --- |
| Case 3/C, having two children. |

1. “Jamming the 25 tree under the 75 tree” will result in a tree that has what undesirable property?

|  |
| --- |
| It will have the undesirable property of: could find a proper location for it, but it’s a lot of work, makes it more like a linked list, which is what we want to avoid. |

1. Will we actually remove the **node** containing the value 50? If not, how will we remove 50 from the tree?

|  |
| --- |
| Never get rid of it, get rid of the value itself with something else. Replace it with the next smaller value in the BST. |

1. Mechanically, what path does one follow in order to find the predecessor value in the tree?

|  |
| --- |
| To find the predecessor value in the tree, take one step to the left, and then go as far as you can to the right. Then you will find the next largest value in the tree. |

1. In the third case (removing a value stored in a node with two children), what value should you always use in this class (unless specifically instructed otherwise)?

|  |
| --- |
| You should use the value that is the next smallest to the number that you are replacing. |

**VIDEO: BST: Remove By Hand Exercise**

1. What is the basic goal of this video?

|  |
| --- |
| Demonstrate how to solve the removal problems by hand for BST. |

1. Show the tree that will result from adding the values 10, 4, 6, 8, 2, 1  
   (Make sure that you understand why the tree looks like this, and not any other possible tree that contains all of these values)

|  |
| --- |
| 10 |  4  ^ |\  2 6 | \ 1 8 |

1. For this exercise, should you simply show the tree that results from removing the target value, or should you actually walk through the logic involved in removing the target value?

|  |
| --- |
| You should also walk through the logic involved in removing the target value. |

**VIDEO: BST: Remove (Overview)**

1. What is the first case we consider, when removing a value from a BST?

|  |
| --- |
| If the target node has zero children |

1. For the first case: In terms of searching for the target value, where do we have to stop in order to be able to remove the target value from the tree?

|  |
| --- |
| At the very end of the tree, where both the left and right values are equal to null |

1. What is the second case?

|  |
| --- |
| The second case is that a BST has one child node |

1. In the second case, what's the (basic, intuitive) first stage in removing the target value?

|  |
| --- |
| Splicing 1 out of the tree by assigning the upper value to the child node of the value that you want to remove. |

1. Once we've found the target value (in a specific node), how will we remove that value from the tree (intuitively)?

|  |
| --- |
| Splicing out the connection to the upper value that the node links to, and changing the upper node to link with the child node of the value that you intend to remove. |

1. Why is it safe to remove that node in the way you said in the prior question?

|  |
| --- |
| Because the values will all be less than the upper value. |

1. What defines the third case?

|  |
| --- |
| The target having two child nodes |

(The following questions all apply to the third case)

1. Instead of removing the NODE containing target value, what are we going to do instead?

|  |
| --- |
| Change the value of 5 to be the next largest value in the tree |

1. In order to do what we said we'd do in the prior question, which value are we going to find in the tree?

|  |
| --- |
| The next largest value in the tree |

1. What path through the tree will we follow in order to find the value described in the prior question?

|  |
| --- |
| Go left once, and then go down as far as you can down to the right of the node |

1. In the example given, do we know if the node with the value of "3" has a left child?

|  |
| --- |
| We know that the value of 3 does not have a left child |

1. What will we do with the value 3? What happened to the node that used to contain 5?

|  |
| --- |
| We will overwrite the value of 5 with 3, and remove the node at the end of tree than contained 3 originally. |

1. What do we do with the node that originally contained 3?

|  |
| --- |
| It will be removed |

1. In this context, what does the word "predecessor" mean?

|  |
| --- |
| The word predecessor refers to the next smallest node beneath the node that you are changing |

1. Why is it easier (and safe) to remove the node that originally contained the value 3?

|  |
| --- |
| Because there are no children of 3 |

1. How do we know that we haven't destroyed the search property on the node that originally contained the value 5?  
   (Specifically – how do you know that the Find method will still correctly find values greater than 5? How do you know that Find will correctly find values less than 5?

|  |
| --- |
| Because when we go one to left, and all the way down to the right, we find the number that is larger than all other numbers in the left of the tree, but still smaller than the one in the right of the tree. This way the same logic applies for searching as did with number 5. |

1. In terms of the predecessor node, are there any guarantees about how many children it has?

|  |
| --- |
| There are no guarantees with how many children it has. |

1. In terms of removing the predecessor node, what was significant about the situation where we wanted to remove the value 1 from the tree? How did it differ from the situation where we were removing the value 5?

|  |
| --- |
| Take a step to the left 🡪 get 0  Then while the right pointer is not null, step the right, there is no right child, so it stops. Code must accommodate for no step to the right. |

1. Instead of using the predecessor value, what other value could we use?   
   (Make sure that you know BOTH what the 'name' of that type of node is, and that you can describe what value that is)

|  |
| --- |
| Instead we can use the next larger value, the smallest value larger than 5. Step to the right, then as far left as you can to find the next smallest number. Perform the same overwrite. |

1. In this class, unless the instructor says otherwise, which node should you remove from the tree in order to correctly do the third case of BST.Remove?

|  |
| --- |
| You should remove the node: the predecessor |

1. Assuming that the tree is balanced, what is the running time (in Big Oh notation) of BST.Remove?

|  |
| --- |
| The running time when balanced is O(log\_2N) |

1. How do computer scientists write log2 (at least when talking about Big Oh notation)?  
   (Will mathematicians generally, and your math teacher specifically, accept this as valid notation?)

|  |
| --- |
| O(log\_N), mathematicians won’t accept it. |

1. What is the worst case structure of the tree when we're doing a remove?

|  |
| --- |
| A linked-list like structure |

1. In the worst case, what is the running time (in Big Oh notation) of doing a BST.Remove?

|  |
| --- |
| O(N) |

1. Does the BST.Remove actively balance the tree as it removes values?  
   (What are the names of two other binary tree structures that do actively balance the tree)?

|  |
| --- |
| It technically does not balance anything  1. AVL  2. Red-black |

**VIDEO: QuickSort Algorithm**

1. What can you use QuickSort on?

|  |
| --- |
| Quicksort can be used on: arrays, multiple arrays, lists, etc. |

1. Is Quicksort always faster than BubbleSort? If not, when will/won't it be faster?

|  |
| --- |
| Quicksort is often faster than bubbleSort  It will not be faster in a worst case scenario  Quicksort takes longer if the array is already sorted |

1. Quickly summarize how BubbleSort works

|  |
| --- |
| A sorting algorithm that repeatedly steps through the list to be sorted, comparing each adjacent pair of items and then swapping them if they are in the wrong order. |

1. What is the running time of BubbleSort (in Big Oh notation)?

|  |
| --- |
| The running time for Bubblesort is O(N) |

1. How is Quicksort fundamentally different from BubbleSort?

|  |
| --- |
| Quicksort partitions items so that all those items with less than the select pivot/key are at the front of the array. All those with greater values are at the back of the array. The pivot number is put in the middle.  Unlike Bubblesort, Quicksort then sorts the reduced lists separately, when it is done, the whole list will be in order. |

1. What is the pivot value? Where will we choose the pivot value from?

|  |
| --- |
| The pivot value is a value somewhere in between the largest values in the array, and the smallest. It isn’t always exactly in the middle though 🡪 Pick something randomly, pick the first one in the array |

1. Each time QSort gets called, how is it going to arrange the elements of the subsection of the array which QSort got called on?

|  |
| --- |
| Each time qsort get called, it’s going to arrange the elements of each subsection depending on whether they are larger than the pivot value, or less than the pivot value (right and left halves)  Work indeces inwards from both ends of the array.  Start from left and look for first element greater than the pivot  Start from the right, look for first element less than pivot  Swap the two items, they will now be on the correct ends of the array, repeat until they meet |

1. Is there any guarantee that the half the values will be placed to the left of the pivot, and half to the right of the pivot?

|  |
| --- |
| No, there may be more that are greater than there are than less of the pivot value |

1. Which function is responsible for choosing the pivot value? Does that function do anything else?

|  |
| --- |
| The pivotIndex function |

1. What value does Partition return?

|  |
| --- |
| The value that the partition returns is always the last element in the array |

1. Each time QSort gets called, where is the pivot value placed, relative to it's final, sorted location?

|  |
| --- |
| At the front of the array |

1. Which value will we choose for the pivot value?

|  |
| --- |
| 40 (the first value in the array as the pivot value) |

1. How will we figure out how to swap a too-large element with a too-small element?

|  |
| --- |
| We will move to the next value in the array to the right of the pivot value that is larger than the pivot value. We will then compare it to the right side of the array, and move left until we find a value smaller than the pivot value. The larger value and the smaller value will then be swapped. |

1. What is the running time (in Big Oh notation) of the partition function?

|  |
| --- |
| O(NLog\_2 \* N) |

1. In the partition function, how do we determine if an element on the left is too large? How do we determine if an element on the right is too small?

|  |
| --- |
| We compare it to the pivot number, if the number on the left is too large, we will look at the right side of the array until we find a number that is smaller than the pivot value, then we will swap the too large number with the smaller number on the other side. |

1. When does the partition function stop walking the left & right indices?

|  |
| --- |
| When the search for the left and right indices overlap |

1. Once we've stopped walking the left & right indices, where do we move the pivot value to?

|  |
| --- |
| We move the pivot value with the number in the middle of the partition function that is smaller than the pivot value. Once swapped, the pivot value will be in the correct sorted position in the array. The new pivot value becomes the next smallest value in the array |

1. After the Partition function has rearranged the array, what function will be called next?

|  |
| --- |
| After rearranging the array, the function that will be called next is QSort |

**VIDEO: QuickSort Time / Space**

1. After calling Partition, where does the pivot element end up (and why)?

|  |
| --- |
| After calling partition, the pivot element ends up in between, half on right, half on left. The array is sorted so that half are larger than the pivot element, and half are smaller. |

1. Hopefully, how large should the left and right halves be after calling partition?

|  |
| --- |
| Ideally they would each represent a half of all elements in the array. |

1. Given an array of size N, how deep should the function call tree (of QSort invocations) be? (Assuming "nice" splits from the Partition function)

|  |
| --- |
| N/4 🡪 log\_2(N) levels deep |

1. How much time will Partition take, in total, to partition each level of the call tree?

|  |
| --- |
| n/2 + n/2 – technically (n-1)/2 = O(N) TECHNICALLY a little less?? Depth \* amount of work per level = O(N \* log\_2N) |

1. What is the running time, in Big Oh notation, of QuickSort (assuming "average" splits from the Partition function)

|  |
| --- |
| O(N\*log\_2(n)) |

1. How much space will the QuickSort algorithm require (in the "average" case)?

|  |
| --- |
| O(log\_2N) |

1. What is the worst possible array for QuickSort to be called on (where "worst" means most time/space – consuming)

|  |
| --- |
| The worst possible case: already sorted array you are trying to quicksort |

**VIDEO: QuickSort Worst Case**

1. When the array is sorted, which element will Partition choose as the pivot element?

|  |
| --- |
| We will choose the first element in the array |

1. In the worst case, how many times will QuickSort be recursively called?

|  |
| --- |
| N times |

1. In Big Oh notation, what is the worst-case **running time** of QuickSort?

|  |
| --- |
| O(N^2) |

1. In Big Oh notation, what is the worst-case **space required** by QuickSort?

|  |
| --- |
| O(N) |

1. In terms of optimizing Quicksort: When the size of a slice of the array gets down to a small number (such as 5 – 10 items), what will some implementations of Quicksort do *instead* of continuing to recursively call itself? Why is this more efficient?

|  |
| --- |
| Will do: Say if span left and the right is 10 or 20, just call bubble sort on the remaining 10 things  More efficient because: reduces the number of recursive calls. |

1. If you were to pick the pivot element randomly, what is something that you'll have to watch out for / fix (in the implementation discussed in the video)

|  |
| --- |
| If you pick the pivot element randomly, watch out for 🡪 make sure things don’t get to the edge, accidently go beyond the bounds of the array. |