

5

1. Sorting Specialists

- a. Radix Sort
- b. Bucket Sort
- c. Merge Sort
- d. Bucket Sort Questions

- i. Explain why Bucket Sort has the constraint of only using numbers between 0 and 1.

- 1. This constraint occurs because Bucket sort takes the ~~leading digit~~ of a number and places it in a bucket. This digit is made from multiplying the number by 10, and getting the value of the ones digit. If Bucket sort accepted numbers of all digits, then numbers like 0.7 and 70 would be put in the same bucket, even though they are different classes of integers.

- ii. How can we modify the algorithm in class to work on numbers greater than one?

- 1. We can modify bucketsort to accept any numbers by padding the numbers to be the same amount of digits, so they would all have different leading digits. This sort would be similar to Radix Sort.

2. More Sorting!

- a. Given the code for Radix Sort, implement the sortDigit() function.
 - i. (sortDigit() is just countSort())
 - ii.

```
def sortDigit(arr, exp, n):
    output = []
    countArr = [0]*10

    for i in range(n):
        numInArray = arr[i]//exp%10
        count[numInArray] += 1

    for i in range(1, 10):
        count[i] += count[i-1]

    for i in reversed(range(n)):
        numInArray = arr[i]//exp%10
        output[countArr[numInArray] - 1] = arr[i]
        countArr[numInArray] -= 1

    arr = output.copy()
```

b. Best/Average/Worst-Case for QuickSort:

- i. Let n = number of items in array:
- ii. Best Case: $O(n \log n)$
- iii. Average Case: $O(n \log n)$
- iv. Worst Case: $O(n^2)$

$\times 3$

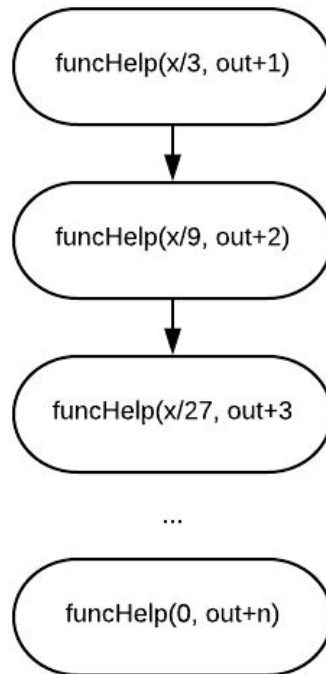
✓

Space?

u

3. Recursion All the Way Down

a. Draw a recursion tree for funcHelp



✓

i.

\times

b. What is the tightest Asymptotic Complexity of funcHelp?

$\lg(b)$

\times

- i. In the func function, funcHelp is called on each element however many times the length of list A is. For example, if list A had a length of 5 and list B had a length of 10, funcHelp would be called on just the first 5 elements of the list. Along with this, funcHelp calls itself the same amount of times as 1/3rd of the length of list B. Because of both of these, the final runtime of this function is **$O(a + b/3)$, where a is the length of the list a and b is the length of the list b .**

\times

c. What is the space complexity of this function?

- i. Every time funcHelp returns back to the fun function it adds onto a list. funcHelp is called the same amount of times as the length of the list A.

\times

Because of this, the space complexity is **$O(a)$, where a is the length of list a .**

\times

$a \lg(b)$

6

4. Fixing Binary Search

- a. Matthieu's `binarySearch` function takes two parameters, a value to search for and the root of a node. It then sets a temporary pointer to point to the root. The function then enters a while loop that continues to run until the temp is not null. If the value is greater than the current value of the node, Matthieu's code sets the left child to the temp. If the value is less than the current node, Matthieu's code sets the right subtree as the temp.. At the end, the code will always return -1 since that is the line after the while loop that will always execute. *-1: int. loop*
- b. This code returns -1 if it doesn't find the value and returns the value if it's found.

```

struct BSTNode {
    int data;
    BSTNode* left;
    BSTNode* right;
}

int binarySearch(int value, BSTNode* root){
    BSTNode* temp = root;
    while (temp != nullptr) {
        if (value > temp->data) {
            temp = temp->right;
        } elif ( value < temp->data) {
            temp = temp->left;
        } else {
            return value;
        }
    }

    return -1;
}

```

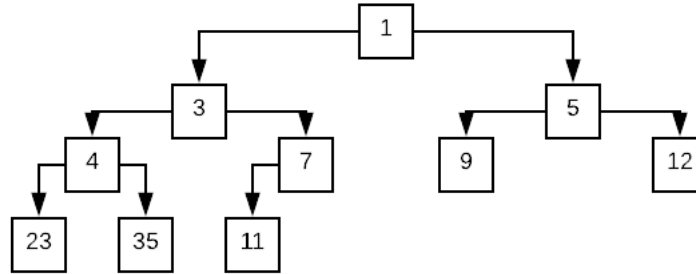
- c. It's not possible to do a recurrence relation for this code as it's iterative and not recursive.

10

5. Heaps of Fun

a. Is [1, 3, 5, 4, 7, 9, 12, 23, 35, 11] a heap? If not, explain why.

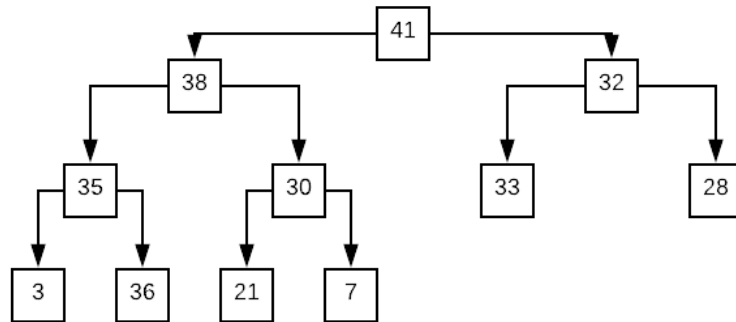
- i. This is a min heap. Each root for each subtree is the minimum value of all of its children.



ii.

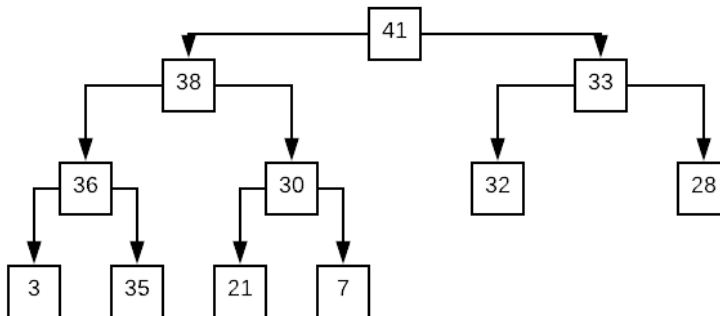
b. Is [41, 38, 32, 35, 30, 33, 28, 3, 36, 21, 7] a heap? If not, explain why.

- i. The way this array is given is not a heap, although it attempts to be a max heap. This is the tree representation:



1.

- ii. In order to make this a max heap, the array must be changed to [41, 38, 33, 36, 30, 32, 28, 3, 35, 21, 7], here is the tree representation:



1.

Ameer Hassan - CS435: 004

c. Write Pseudocode for Heapsort

```
currHeap = Heap()
inputArr = [1, 65, 4, 3, 63, 22, 68, 99, 23]
outputArr = []

for num in inputArr: # Insert all the elements in the input into the Heap
    currHeap.insert(num) ✓

for i in range(len(inputArr)): # Extract the maximum of the heap
    outputArr.append(currHeap.extractMax()) ✓
```

18

6. Interview Style Question 1

a. Repeat the Question

- i. Write a function that returns an integer that is the largest suffix of another number.

b. Questions about edge cases

- i. This function assumes that if there is no suffix, then it returns None.

c. Think of a solution

- i. Insert each element reversed into a trie. If an element is in the Trie reversed, then that element is the suffix of another previous element that was in the Trie.

d. Code

```
def findIntAtEnd(inp):
    # Create a tree and a variable to store the largest integer suffix.
    reversedTrie = Trie()
    largestSuffix = None

    # Sort the list so it is in descending order
    inp.sort(reverse=True)

    # Iterate through all the
    for elem in inp:
        # Reverse the current element
        reversedElem = reversed(elem)

        # Check if the element exists in the trie that holds reversed
        elements
        if(reversedTrie.search(reversedElem)):
            # If there is no stored suffix, store the current suffix
            if (largestSuffix is None):
                largestSuffix = reversedElem
            # If there is a stored suffix, replace it only if it's
            larger
            else:
                if (len(reversedElem) > len(largestSuffix)):
                    largestSuffix = elem

        insertSubstrings(reversedTrie, reversedElem)

    return largestSuffix

# Function that inserts a string and all of its substrings into a trie.
# This makes every node of the trie a leaf.
def insertSubstrings(currTrie, currStr):
    for i in range(1, len(currStr)+1):
        currTrie.add(currStr[:i])
```

what if the largest suffix is inserted last? -5

- e. Test Cases
 - i. Through various test cases, this solution seems to work.
- f. Iterate
 - i. Could come up with a way to modify the add of the trie to make every node a leaf, instead of having a dedicated method for it. ✓
- Time Complexity:
 - Time Complexity (n =size of list, m =maximum num of digits) ✗
 - $O(n+m)$ - You are iterating through the whole list once, and you are iterating through the digits of each number once. ✗ $O(n \times m)$
 - Space Complexity (n is the number of unique numbers, m is the number of digits(10))
 - $O(n+m)$ ✗ $O(n \times m)$

✗ 3

25

7. Put it Together

- a. Repeat the Question ✓
 - i. Given two binary search trees, create a list that contains all of their values in ascending order.
- b. Questions about edge cases ✓
 - i. Assuming that both trees have at least one item in them, assuming if findNext is called on an element that is the maximum of the tree it returns None.
- c. Think of a solution ✓
 - i. Have two variables that store the minimum of each tree. Find the minimum of the two variables, append it to the output array, find the next largest value of that tree, continue the loop.
- d. Code on next page.

Don't
do a)
and b)
on the
exam!

Ameer Hassan - CS435: 004

```
def printAllNums(first, second):
    # Two variables that always hold the minimum value of the tree.
    # Declaring the output list
    currFirst = first.findMin()
    currSecond = second.findMin()
    outputList = []

    # While both trees are populated:
    while (currFirst is not None and currSecond is not None):
        # If the first minimum is less than the second, append
        # the value to the output list and find the next value
        # If it's vice versa, append the second. If they're the same,
        # Add them both and get the next value.
        if (currFirst < currSecond):
            outputList.append(currFirst)
            currFirst = first.findNext(currFirst)
        elif (currFirst > currSecond):
            outputList.append(currSecond)
            currSecond = second.findNext(currSecond)
        else:
            outputList.append(currFirst)
            outputList.append(currSecond)
            currFirst = first.findNext(currFirst)
            currSecond = second.findNext(currSecond)

    # Add the rest of the remaining tree that is populated.
    if(currFirst is None):
        while(currSecond is not None):
            outputList.append(currSecond)
            currSecond = second.findNext(currSecond)
    elif(currSecond is None):
        while(currFirst is not None):
            outputList.append(currFirst)
            currFirst = first.findNext(currFirst)

    return outputList
```

Ameer Hassan - CS435: 004

- e. Test Cases
 - i. This method seems to work with any two trees, given that the functions are provided. ✓
- f. Iterate
 - i. Could maybe come up with a way that does not even need the two variables in the beginning, since space is important in this problem. ✓
- Complexities
 - **Time Complexity:** Let n = number of nodes in first tree. Let m = number of nodes in second tree. ✓
 - $O(n + m)$. Since we are going through each node in each tree once, we have to linearly go through each node once.
 - **Space Complexity:** ✓
 - $O(n + m)$: The output list holds all of the nodes in each tree. The space needed to compute this array is constant, the output array is the one with size $n + m$.

220

1. Almost BST
 - a. Repeat the Question
 - i. Given a tree, see if it can be a Binary Search Tree if at most one node is out of place. ✓
 - b. Edge Cases
 - i. Assuming the input has at least one node in it ✓
 - c. Solution
 - i. Generate an in-order traversal of the tree. If there is at most one element that is not in sorted position, return true. If there are more than one elements that are not in sorted position, return false. ✓

Ameer Hassan - CS435: 004

d. Code

```
class Node():
    def __init__(self, value):
        self.value = value
        self.right = None
        self.left = None

class BST():
    def __init__(self):
        self.root = None
        self.inOrderList = []


    def inOrder(self):
        self.inOrderList = []
        self.inOrderHelper(self.root)

    def inOrderHelper(self, currNode):
        if currNode != None:
            self.inOrderHelper(currNode.left)
            self.inOrderList.append(currNode.value)
            self.inOrderHelper(currNode.right)

    def almostBST(self):
        almostBST = 1
        self.inOrder()

        for idx in range(len(self.inOrderList)-1):
            if (self.inOrderList[idx] < self.inOrderList[idx+1]):
                continue
            else:
                if (almostBST is 0): # If there is already an out of
place node, return F.
                    return False
                almostBST = 0

        return True
```



e. Test Cases

f. Iterate

time $\frac{1}{2}$ space?
+0