

2201-COL106 Minor

Anish Banerjee

TOTAL POINTS

12 / 40

QUESTION 1

1 0 / 5

- ✓ **- 5 pts** Wrong answer or answered "Yes"
 - **4 pts** Answered "No" without proper example or justification
 - **1 pts** Answered "No" with incomplete explanation or example
 - **0 pts** Answered "No" with proper example and justification

QUESTION 2

2 5 / 5

- + **0 pts** Incorrect / Not Attempted
- ✓ + **1.25 pts** Correct while loop condition
- ✓ + **1.25 pts** Correct update statement in if block
- ✓ + **1.25 pts** Correct termination condition
- ✓ + **1.25 pts** Correct time complexity
- + **0 pts** Minor inaccuracies

QUESTION 3

3 2 / 5

- + **0 pts** Incorrect
- + **5 pts** Correct
- **1 pts** Incorrect complexity
- **4 pts** Inefficient solution
- **0.5 pts** For incorrect 1st blank
- ✓ + **0.5 pts** For correct 1st blank
- **1.5 pts** For incorrect 2nd blank
- ✓ + **1.5 pts** For correct 2nd blank
- **2 pts** For incorrect 3rd blank
- + **2 pts** For correct 3rd blank

QUESTION 4

10 pts

4.1 5 / 5

- + **0 pts** Incorrect / Not Attempted
- + **1 pts** Partially Correct Ideas: Complete binary tree Assumption upto level d / All leaves with same shallowest depth etc.
- ✓ + **3 pts** Correct Expression
- ✓ + **2 pts** Correct Explanation
- + **1 pts** Partially Correct Explanation

4.2 0 / 5

- ✓ + **0 pts** Incorrect / Not Attempted
- + **2 pts** Expression / Value off by a constant
- + **2 pts** Correct Expression / Value
- + **3 pts** Correct Explanation

QUESTION 5

5 0 / 15

- **0 pts** Correct code
- **2 pts** Code is slightly incorrect
- **5 pts** Right idea, but many mistakes in code
- ✓ - **10 pts** Incorrect /missing code
- ✓ - **5 pts** Incorrect /missing proof of correctness
 - **0 pts** Correct proof of correctness
 - **3 pts** Proof of correctness partially correct

Name: ANSH BANERJEE

Entry number: 2021CS10134

COL106

Minor Exam

Duration: 1 hour

Read the following instructions before you begin writing.

1. Keep a pen, your identity card, and optionally a water bottle with you. Keep everything else away from you, at the place specified by the invigilators.
 2. Write your entry number and name on every page. (Sheets will be separated prior to grading.)
 3. Answer only in the designated space. Think before you use this space. No additional space will be provided for writing answers. Use the last blank sheet for rough work, if needed. Do not separate sheets from one another.
 4. No clarifications will be given during the exams. If something is unclear or ambiguous, make reasonable assumptions and state them clearly. The instructors reserve the right to decide whether your assumptions were indeed reasonable.
-
1. (5 points) Suppose $f, g : \mathbb{N} \rightarrow \mathbb{N}$ are functions such that $f(n)$ is $O(g(n))$. Does this necessarily imply that $2^{f(n)}$ is $O(2^{g(n)})$? Justify your answer, ideally in at most three sentences.

$$f(n) = O(g(n)) \Rightarrow \exists c \exists n_0 : \forall n > n_0 : f(n) \leq c g(n)$$

As 2^x is an increasing function, we have

$$\Rightarrow 2^{f(n)} \leq 2^{c g(n)}$$

$$\Rightarrow 2^{f(n)} \leq 2^c \cdot 2^{g(n)}$$

$$\Rightarrow 2^{f(n)} \leq c' 2^{g(n)}$$

$$\text{Thus } 2^{f(n)} \text{ is } O(2^{g(n)})$$

Name: ANISH BANERJEEEntry number: 2021CS10134

COL106

Minor Exam

Duration: 1 hour

2. (5 points) Each node of a singly-linked list has a color, which is either black or white. We want to determine whether the number of black nodes equals the number of white nodes, and we want to do this without using a counter to count the nodes in the list. List L has the following methods.

- `first()`: returns the first node of L.
- `after(v)`: returns the node after node v in L.
- `isempty()`: returns True if L is empty, and False otherwise.
- `isLast(v)`: returns True if v is the last node of L, and False otherwise.
- `deletefirst()`: deletes the first node of L.
- `deleteafter(v)`: deletes the node after v in L.

The class Node has a method `color()` so that the call `v.color()` takes $O(1)$ time and returns the color ("Black" or "White") of node v.

Complete the following pseudocode of an algorithm which takes a singly-linked list L, and returns True if the number of black and white nodes in L is equal, and False otherwise.

`Colorbalanced(L):`

`q = L.first()`

 while not(`L.isLast(q)`) do
 if `L.first().color == L.after(q).color`:

`q =` `L.after(q)``-----`

 else:

`L.deleteafter(q)`

 if `L.first() == q`:

`q = L.after(q)`

`L.deletefirst()`

 if `L.isempty()`:
 return True

 return True

 # End of while loop

 return False

Write down a function $f(n)$ such that the worst-case running time of the above method on a linked list of size n is $\Theta(f(n))$.

n

Name: ANISH BANERJEEEntry number: 2021CS10134

COL106

Minor Exam

Duration: 1 hour

3. (5 points) Consider a modified implementation of the `BinaryTreeNode` class so that an object has, in addition to the usual attributes `key`, `parent`, `left`, `right`, an additional attribute `size`, where `p.size` equals the number of nodes in the subtree rooted at node `p`, including `p` itself. A `BinarySearchTree` object has a single attribute: `root`, a reference to an object of our modified `BinaryTreeNode` class. Your goal is to add an accessor method `countgeq` to the `BinarySearchTree` class so that if `T` is a `BinarySearchTree` object, then the call `T.countgeq(x)` returns the number of keys in `T` that are greater than or equal to `x`. Complete the following code snippet into an **efficient** implementation of `countgeq` by filling the blanks.

```
def countgeq(self, x):
    return self.countgeq_helper(self.root, x)

def countgeq_helper(self, p, x):
    # Counts the number of keys in the subtree of self rooted at p
    # that are >= x.

    if p is None:
        return 0
    if p.key < x:
        return countgeq_helper(p.right, x)
    else:
        return p.size
```

Write down a function $f(h)$ such that the worst-case running time of the above method is $\Theta(f(h))$, where h is the height of the binary search tree.

h

Name: ANISH BANERJEEEntry number: 2021CS10134

COL106

Minor Exam

Duration: 1 hour

4. Recall that the depth of a node in a binary tree is the length of the path from the root to that node. For instance, the root has depth 0, its children have depth 1, and so on. Consider an AVL tree in which the shallowest leaf (leaf with least depth) has depth equal to the trailing five-digit number in your entry number (eg. 34567 in 2021AB34567). Answer the following questions with (an arithmetic expression which evaluates in $O(1)$ time to) the correct numerical answer (no recurrences), and a short explanation.

- (a) (5 points) What is the minimum possible number of nodes in the tree?


We will have minimum possible nodes when all the leaves are at the same level. In the ~~last~~ last layer, every node has one child. Above it, the tree is complete.

last 5 digits = 10134

Their parents have one child

$$\sum_{i=0}^{10133} 2^i + 2^{10133} = 2^{10134} - 1 + 2^{10133}$$

- (b) (5 points) What is the maximum possible depth of the deepest node in the tree?

 Total leaves in min depth level = 2^{10134}

$$10134 + 2^{10134} - 1 = 10133 + 2^{101}$$

keep increasing by 1 as you move left

5. (15 points) Consider the standard implementation of the MinHeap class where an object has the following two attributes.

- levelorder: a list which stores the elements of the heap in level order
- size: the number of elements in the heap

The implementation supports the following methods.

- MinHeap(): a constructor which returns an empty min-heap
- enqueue(x): adds the element x into the heap
- extractMin(): removes the minimum element from the heap and returns that element
- isEmpty(): returns True if the heap is empty and False otherwise

Your job is to add an **accessor** method **orderstat** so that if H is a MinHeap object with integer keys and k is a natural number, the call **H.orderstat(k)** returns the k'th smallest element of the heap (a.k.a. the k'th *order statistic*).

Here is a useful fact. Python's comparison operators compare tuples position-by-position. For example, the expression ' $(x1, y1) < (x2, y2)$ ' is equivalent to ' $x1 < x2$ or $(x1 == x2 \text{ and } y1 < y2)$ '. This enables us to build min-heaps whose keys are pairs of numbers.

Name: ANISH BANERJEEEntry number: 2021CS10134

COL106

Minor Exam

Duration: 1 hour

Complete the following implementation of `orderstat` by adding (ideally, at most 6) lines of code inside the `for` loop. For nonzero points, your implementation must run in time $O(k \log k)$ (and therefore, independent of n , the number of elements in the `MinHeap` object on which it is invoked).

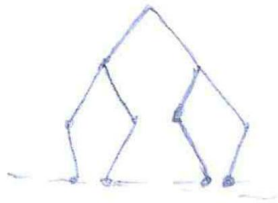
```
def orderstat(self, k):  
    # You may assume  $k \leq \text{self.size}$ .  
    aux = MinHeap()  
    aux.enqueue((self.levelorder[0], 0))  
    for i in range(0, k):  
        (key, pos) = aux.extractMin()
```



```
-----  
-----  
-----  
-----  
-----  
  
# Everything above is *inside* the "for i in range(0, k)" loop.  
# The next statement is *outside* the "for i in range(0, k)" loop.  
return key
```

Using a loop invariant, briefly prove that your implementation is correct.

$$2^{l+1} - 1$$



210134

