Name: _____ **Entry number:** _____

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 $f, g : \mathbb{N} \longrightarrow \mathbb{N}$ 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.

NO. $2n$ is $O(n)$ but 2^{2n} is not $O(2^n)$ because, given any c, N, if we take $n = \max(c, N)$, we have $\frac{2^{2n}}{2^n} = 2^n \geq 2^c > c$.

Name: _____ Entry number: _____

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
        # 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: _____ Entry number: _____

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(self, p.right, x)
    else:
        return 1 + (0 if p.right is None else p.right.size) + countgeq_helper(self, p.left, x)
```

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: _____ Entry number: _____

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?

   Levels $0 \ldots n-1$ must be full, otherwise $\exists$ a leaf or an unbalanced node in those levels.
   $\geq 2^{n-1}$ nodes in level n to avoid a leaf in level $n-1$.
   $\therefore$ Answer: $1+2+\cdots+2^{n-1}+2^{n-1} = 2^n+2^{n-1}-1$.

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

   Answer: $2n$. As we walk from the shallowest leaf to root, the height of the subtree rooted at the current node increases by ≤ 2, and an increase of 2 can be actually achieved.

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 and y1<y2)`'. This enables us to build min-heaps whose keys are pairs of numbers.

Name: _____ **Entry number:** _____

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 <= self.size.
    aux = MinHeap()
    aux.enqueue((self.levelorder[0],0))
    for i in range(0, k):
        (key, pos) = aux.extractMin()
        pos1 = 2*pos+1
        if pos1 < self.size:
            aux.enqueue((self.levelorder[pos1], pos1))
        pos2 = 2*pos+2
        if pos2 < self.size:
            aux.enqueue((self.levelorder[pos2], pos2))
        # 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.

Invariant: At the beginning of the for loop, aux contains (x, pos) iff

1. Parent of x in self is among the least $i-1$ elements of self
2. x is not among the least $i-1$ elements of self.
3. self.levelorder[pos] == x.

∴ The value of key in the ith iteration is the ith smallest element of self.

(Parent of the ith smallest element in a min heap is the jth smallest element in the min heap, for some $j < i$.)