DSC 40B - Homework 04

Due: Tuesday, February 2

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope on Tuesday at 11:59 p.m.

Problem 0.

We'd like to hear how things are going. What are we doing that is working? What could we be doing better? If you fill out the Google Form below corresponding to your section, we'll give you two points of extra credit on this homework.

- Dr. Eldridge's section: https://forms.gle/m3YGfWYUaqT2DUc67
- Dr. Wang's section: https://forms.gle/gw57anSKEoQKp2aW6

The form is completely anonymous, so we'll use the honor system to assign credit for this problem. To get your extra credit, simply write in your answer to this question that you filled out the form.

Problem 1. *(Eligible for Redemption)*

What is the *best* case time complexity of `quickselect`? Describe why and provide the input that yields this best case.

Problem 2. *(Eligible for Redemption)*

Determine the worst case time complexities of the following functions under the assumption that the binary search tree being operated is balanced tree of n nodes.

For this problem, assume that we represent a binary search tree node as an object with `left`, `right`, `key`, and `parent` attributes. If `left`, `right`, or `parent` are `None`, it means that the node does not have a left/right child or a parent, respectively.

a) Assume that the input, `root`, is the root of the binary search tree (and is not `None`).

```
def foo(root):
    node = root
    while node is not None:
        parent = node.parent
        node = node.left
    return parent.key
```

b) Assume that the input, `root`, is the root of the binary search tree (and is not `None`).

```
def bar(root):
    if root is not None:
        bar(root.left)
        print(node.key)
        bar(root.right)
```

Problem 3. *(Eligible for Redemption)*

Suppose a binary search tree has been augmented so that each node contains an additional attribute called `size` which contains the number of nodes in the subtree rooted at that node. Complete the following code so that it computes the value of the kth smallest key in the tree, where $k = 1$ is the minimum.

```
def order_statistic(node, k):
    if node.left is None:
        left_size = 0
    if node.right is None:
        right_size = 0

    order = left_size + 1

    if order == k:
        return node.key
    elif order < k:
        return order_statistic(...)
    else:
        return order_statistic(...)
```

Problem 4.

Consider this version of quicksort given below. It is essentially the same as that given in lecture, except that 1) it always uses the last element of the array as the pivot, and 2) it has a `print` statement inserted at a crucial place.

```
def quicksort(arr, start, stop):
    """Sort arr[start:stop] in-place."""
    if stop - start > 1:
        pivot_ix = partition(arr, start, stop, stop-1)
        quicksort(arr, start, pivot_ix)
        quicksort(arr, pivot_ix+1, stop)


def partition(arr, start, stop, pivot_ix):
    def swap(ix_1, ix_2):
        arr[ix_1], arr[ix_2] = arr[ix_2], arr[ix_1]

    pivot = arr[pivot_ix]
    swap(pivot_ix, stop-1)
    middle_barrier = start
    for end_barrier in range(start, stop - 1):
        if arr[end_barrier] < pivot:
            print('hello')
            swap(middle_barrier, end_barrier)
            middle_barrier += 1
        # else:
            # do nothing
    swap(middle_barrier, stop-1)
    return middle_barrier
```

Suppose `arr` is an array of length k with entries `[1, 2, 3, ..., n]`, where n is some large integer. If `quicksort(arr, 0, n)` is run, exactly how many times will `"hello"` be printed to the screen? Your answer should be an expression involving n, and should not involve $\sum$ or $\ldots$. Show your work.