
DSC 40B - Quiz 04

Name:

Student ID:

About the quizzes:

- Quizzes in DSC 40B are *optional* and graded pass/fail.
- A score of 70% or higher earns a “pass” and 1.5 credits toward your final grade.
- If you don’t pass, no credits are earned, but it doesn’t hurt your grade.
- You have 30 minutes to complete the quiz.
- Unfortunately, we can’t answer clarifying questions during the quiz. If you think a question has a bug or is unclear, please let us know in a private post on Campuswire after the quiz, and we’ll take it into account when grading.

Problem 1.

True or False: `quickselect` requires the input array to be sorted in order to function correctly.

- ☐ True
☒ False

Solution: False.

`quickselect` works on unsorted arrays: it partitions the array around a pivot, which puts the pivot in its correct sorted position, and then recurses only on the side that contains the k th smallest element. If the array were already sorted, we wouldn't need `quickselect` at all; we could just look up the element at index $k - 1$ in $\Theta(1)$ time.

This problem was Problem #148 from the Quiz 04 practice problems.

Problem 2.

Recall the `partition` operation from `quickselect`. Which one of the following arrays could be the result of partitioning an array around some pivot?

- ☐ [8, 3, 5, 12, 10, 1]
☐ [4, 2, 7, 6, 9, 8]
☒ [3, 1, 2, 5, 9, 7]
☐ [9, 7, 5, 3, 2, 1]

Solution: [3, 1, 2, 5, 9, 7].

After partitioning, the pivot is in its correct place in the sorted order: everything to its left is smaller, and everything to its right is larger. The other elements don't have to be in sorted order. So an array could be the result of partitioning only if one of its elements is larger than everything to its left and smaller than everything to its right.

In [3, 1, 2, 5, 9, 7], the 5 is such an element: 3, 1, and 2 are smaller than 5, and 9 and 7 are larger. None of the other arrays has such an element. For example, in [4, 2, 7, 6, 9, 8], the 7 is larger than everything to its left, but the 6 to its right is smaller than 7.

This problem was similar to Problem #156 in the Quiz 04 practice problems.

Problem 3.

Consider the function `foo` below.

```
def is_even(x):  
    """Returns True if x is even, False otherwise."""  
    return x % 2 == 0  
  
def foo(numbers):  
    barrier_ix = 0  
    for i in range(len(numbers)):  
        if is_even(numbers[i]):  
            numbers[barrier_ix], numbers[i] = (  
                numbers[i], numbers[barrier_ix]  
            ) # swap  
            barrier_ix += 1
```

Which one of the following is a valid loop invariant for the `for`-loop? That is, which one is true after every iteration, no matter what list `numbers` is?

- ☒ All numbers in `numbers[:barrier_ix]` are even.
- ☐ All numbers in `numbers[:barrier_ix]` are odd.
- ☐ All numbers in `numbers` are even.
- ☐ `numbers` is in sorted order.

Solution: All numbers in `numbers[:barrier_ix]` are even.

This works like the in-place `partition` from lecture, with `barrier_ix` playing the role of the middle barrier and `i` the role of the end barrier. Whenever the loop finds an even number at index `i`, it swaps it to index `barrier_ix` and then moves the barrier one to the right, so the newly found even number ends up just to the left of the barrier. The barrier never moves otherwise, so everything to its left is even.

The other choices are not always true. For example, if `numbers = [4, 2, 1]`, then after the first iteration `numbers[:barrier_ix]` is `[4]`, which is even, not odd. The 1 is never moved, so `numbers` always contains an odd number. And `numbers` stays `[4, 2, 1]`, which isn't sorted: the loop only moves the even numbers to the front, without sorting them.

This problem was similar to Problem #144 in the Quiz 04 practice problems.

Problem 4.

In this problem, n is the number of elements in the input array.

a) What is the **expected** time complexity of `quicksort`?

- ☐ $\Theta(\log n)$ ☐ $\Theta(n)$ ☒ $\Theta(n \log n)$ ☐ $\Theta(n^2)$

b) What is the **worst case** time complexity of `quicksort`?

- ☐ $\Theta(\log n)$ ☐ $\Theta(n)$ ☐ $\Theta(n \log n)$ ☒ $\Theta(n^2)$

Solution: Expected time: $\Theta(n \log n)$. Worst case: $\Theta(n^2)$.

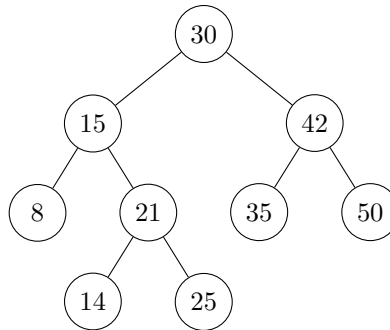
Expected time. Partitioning takes $\Theta(n)$ time. With a random pivot, we expect the pivot to split the array into two pieces of roughly equal size, as if $T(n) = 2T(n/2) + \Theta(n)$. This is the same recurrence as mergesort, and it is $\Theta(n \log n)$.

Worst case. In the worst case, every pivot is the smallest or largest element. Then one of the two recursive calls is on an array of size $n - 1$ and the other is on an empty array, so the time is $T(n) = T(n - 1) + \Theta(n)$, which is $\Theta(n^2)$. This is very unlikely with random pivots.

This was covered in Lecture 07.

Problem 5.

True or False: the tree below is a binary search tree.



☐ True

☒ False

Solution: False.

Remember the BST property: every node in a node's left subtree must be smaller than it, and every node in its right subtree must be larger. This has to be true for *every* node in the tree, so it isn't enough to check the root.

Here, the property holds for the root, 30: everything in its left subtree is smaller than 30, and everything in its right subtree is larger. But it doesn't hold for 15. The node 14 is in 15's right subtree, but 14 is smaller than 15.

This problem was similar to Problem #161 in the Quiz 04 practice problems.

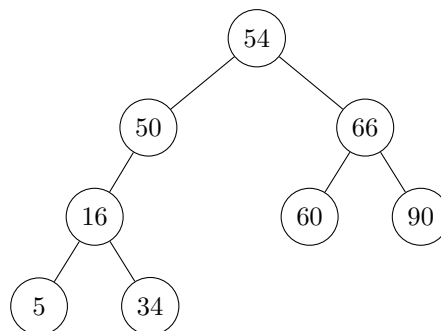
Problem 6.

Recall the query method from lecture. Below, we have added a line that prints "Hello":

```

def query(self, target):
    current_node = self.root
    while current_node is not None:
        print("Hello")
        if current_node.key == target:
            return current_node
        elif current_node.key < target:
            current_node = current_node.right
        else:
            current_node = current_node.left
    return None
  
```

Consider the following binary search tree:



Suppose we do a query for 3 in this tree. How many times is "Hello" printed?

- ☐ 1
 ☐ 2
 ☐ 3
☒ 4
☐ 5

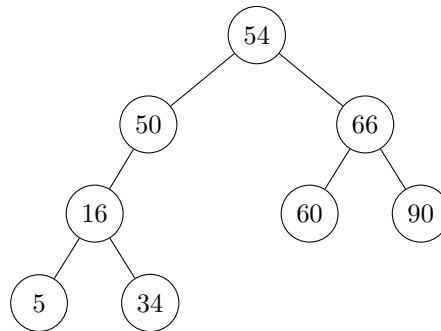
Solution: 4.

"Hello" is printed once for each node the query visits. The query visits 54, then 50, then 16, then 5. Since 3 is smaller than 5, `current_node` becomes 5's left child, which is `None`, so the loop ends and the query returns `None`: 3 is not in the tree.

This problem was similar to Problem #162 in the Quiz 04 practice problems.

Problem 7.

Consider the same binary search tree as in the previous problem:



Where should we insert 49?

- ☐ the left child of 50
☐ the right child of 16
☐ the left child of 60
☒ the right child of 34
☐ the left child of 34

Solution: The right child of 34.

Starting at the root: 49 is smaller than 54, so we go left to 50; 49 is smaller than 50, so we go left to 16; 49 is larger than 16, so we go right to 34; and 49 is larger than 34, and 34 has no right child, so 49 becomes the right child of 34.

This problem was Problem #163 from the Quiz 04 practice problems.

Problem 8.

Suppose `bst` is an empty binary search tree, and that the code below inserts n keys into it using the `insert` method from lecture.

```

for i in range(n):
    bst.insert(i)
  
```

What is the time complexity of this code, as a function of n ?

- ☐ $\Theta(n)$
☐ $\Theta(n \log n)$
☒ $\Theta(n^2)$
☐ $\Theta(n^2 \log n)$

Solution: $\Theta(n^2)$.

The keys $0, 1, 2, \dots, n-1$ are inserted in sorted order, so each new key is larger than every key already in the tree, and it becomes the right child of the previous key. The tree is very unbalanced: it is a single chain of right children, like a linked list.

Inserting takes $\Theta(h)$ time, where h is the height of the tree. When i is inserted, the tree is a chain of i nodes, and the insertion walks past all of them to reach the bottom. So inserting i takes $\Theta(i)$ time, and the total time is proportional to

$$\underbrace{0}_{\text{time to insert node 1}} + \underbrace{1}_{\text{time to insert node 2}} + \underbrace{2}_{\text{time to insert node 3}} + \dots + \underbrace{(n-1)}_{\text{time to insert node } n} = \frac{n(n-1)}{2} = \Theta(n^2).$$

This problem was similar to Problem #172 in the Quiz 04 practice problems.

Problem 9.

True or False: in **any** binary search tree with n nodes, the **query** method from lecture takes $\Theta(\log n)$ time in the worst case.

- ☐ True
- ☒ False

Solution: False.

query takes $\Theta(h)$ time in the worst case, where h is the height of the tree. The height is about $\log n$ only if the tree is balanced. If the tree is very unbalanced, as when the keys were inserted in sorted order, then its height is about n , and a query can take $\Theta(n)$ time. So the worst case is $\Theta(\log n)$ only when we know the tree is balanced.

This problem was similar to Problem #172 in the Quiz 04 practice problems.

Problem 10.

Suppose **quickselect** is used to find the **minimum** of an array of n distinct numbers (that is, $k = 1$). Which pivot would be the **worst** choice for the first call, in the sense that it leaves the largest subproblem for the next recursive call?

- ☐ The minimum.
- ☐ The second smallest number.
- ☐ The median.
- ☒ The maximum.

Solution: The maximum.

After partitioning, **quickselect** recurses only on the side that contains the element it's looking for. The minimum is always to the left of the pivot (unless it is the pivot), so the next call is on everything smaller than the pivot. If the pivot is the maximum, that's everything except the pivot: a subproblem of size $n-1$, the largest possible. This is the bad case from lecture that, if it happened every time, would make **quickselect** take $\Theta(n^2)$ time.

If the pivot were the minimum, **quickselect** would be done right away, and the second smallest number or the median would leave only 1 or about $n/2$ numbers to search.

This problem was similar to Problem #160 in the Quiz 04 practice problems.