


# My grades for csc148\_midterm2

Q1

1

Q2

1



45F82AA2-6145-4998-A310-AB115CA1559  
csc148\_midterm2-2week  
#630 Page 2 of 16

1. (1 pt) Runtime complexity

The expected/average time complexity of finding an item in a Binary Search Tree is:

Circle ALL statements that apply.

- A.  $O(1)$
- B.  $O(n)$
- C.  $O(n^2)$
- ☒ D.  $O(\log(n))$
- E.  $O(n \log(n))$
- F.  $O(n^{**2})$
- G.  $O(n!)$
- H.  $O(e^{**}(n!))$
- I. All of the above.
- J. None of the above.

2. (1 pt) Runtime complexity

Given our current understanding of the complexity of finding an item in a Binary Search Tree:

Circle ALL statements that apply.

- A.  $O(1)$
- B.  $O(\log(n))$
- ☒ C.  $O(n)$
- D.  $O(n^2)$
- E.  $O(n \log(n))$
- F.  $O(n^{**2})$
- G.  $O(n!)$
- H.  $O(e^{**}(n!))$
- I. All of the above.
- J. None of the above.

===== Q1 - 1 1

Mark =====

+1/1: **Good Job.**

~gl for the final exam :)

=====

=====

+1/1: **Good Job.**

~gl for the final exam :)

=====

=====

Q3

1

3. (1 pt) List comprehension
- Which statements are true in Python:  
Circle ALL statements that apply.
- ☒ A. All list comprehensions can be converted into multiple lines of Python code involving iterative features such as for loops, if else statements and variable assignments and other features.
  - ☐ B. If a Python program reaches the maximum recursion depth, the best way to proceed is to increase the maximum recursion depth.
  - ☐ C. Recursive functions must have exactly 1 base case and can have any number of recursive calls.
  - ☒ D. If a recursive function has no base case, but has recursive calls, a call to this function will always result in a Exception/Error being raised.
  - ☐ E. Finding the 50th Fibonacci number with a purely recursive algorithm in Python (not using memoization) is an example of a good and efficient algorithm.
  - ☐ F. Recursion is unique to Python.
  - ☐ G. All recursion always leads to the function being in a non-recursive state.

correct, good job! 1

Q4

0



4. (1 pt) Recursion  
Why is partial tracing such a powerful technique for tracing and debugging recursive functions? Select ALL statements that apply.

- ☐ A. Partial tracing allows us to fully trace every single recursive call until it reaches the base case.
- ☐ B. We only use the partial tracing technique when we have lots of time, so that we can diligently trace through all lines within the recursive function, despite its repetitiveness.
- ☐ C. Partial tracing allows us to trace the recursive code without tracing into the recursive call, which means we can assume each call is correct and make sure the rest of the code uses those calls correctly.
- ☒ D. An advantage of using partial tracing means proving that strong induction works at every recursive call so that the base case is never reached.
- ☐ E. The partial tracing technique is useful because we can easily determine if a recursive step is correct without fully tracing the code.
- ☐ F. All of the above.
- ☐ G. None of the above.

Q5

0

5. (1 pt) Binary Search Tree  
What would the following code do:  

```
self._root = self._left._root  
self._left = self._left._left  
self._right = self._left._right
```

  
Select ALL statements that apply.

- ☐ A. Replace this Binary Search Tree with its left subtree.
- ☒ B. Replace the root with the left subtree to this Binary Search Tree.
- ☐ C. Remove this Binary Search Tree's original right subtree.
- ☒ D. Remove the Binary Search Tree's original root node.
- ☐ E. Remove this Binary Search Tree's original left subtree.
- ☐ F. All of the above.
- ☐ G. None of the above.

The correct answers are C and E

The correct answers are A, C, and D

Q6

0


4092P623-4855-4850-0000-CSC01796811  
cas148\_p10fscm0-00003  
#630 Page 5 of 16

**6. (1 pt) Freebie**

Please read the entire question. What is the name of one of the professors/instructors (not TAs) for this CSC148 course? :)

Select only 1 correct answer for full marks, even though multiple answers are correct; selecting more than 1 correct answer will result in 0 points for this question.

- A. Rutwah Engineer
- B. Rutwa Engineerli
- C. Rutwa Engineer
- D. Ahmed Ashraf
- E. Ahmed Ashraf
- F. Ahmed Ashraf
- G. ☒ Andy Bergen
- H. Andi Bergen
- I. Andi Bergen
- J. All of the above.
- K. None of the above.



993730897-9648-4870-8070-5A4F3E7548E3

csu168\_m0ttam2-jmwh

#630 Page 6 of 16

7. (4 pts) Abstract Data Types

Implement the enqueue method of the custom Queue class below. Read all doctstrings carefully.

Restrictions (not following these restrictions will result in zero marks for this question):

- The Queue must not have any other methods aside from those you must implement. The Stack interface is provided in the aid sheet (nothing else can be assumed or modified in it).
- You may use new variables and the boolean object given in the code (e.g. `found`).
- You MUST NOT assume that ANY other method exists for Queue.
- You MUST NOT define any helper functions or methods.
- You MUST NOT create any new objects other than Stacks.
- Your code MUST NOT crash by raising an EmptyStackError.


```
class Queue:
    """
    == Private Attributes ==
    _items: stores the elements of this Queue.
    The top of the Stack represents the back of the Queue.
    The bottom of the Stack represents the front of the Queue.
    """
    _items: Stack
    def __init__(self) -> None:
        self._items = Stack()

    def enqueue(self, item: Any) -> None:
        """
        Add the object referenced by <item> to the back of
        this Queue, if and only if the <item> is unique.
        That is, if the <item> already exists in the Queue do nothing.
        """
        found = False
```

On the NEXT page implement enqueue.

Q7

3

48020373-8064-4861-9167-88F0158E1212  
 cas148\_milestone3-2024s3  
 #630 Page 7 of 16
 


**TODO:** In the space below, implement the enqueue method here (you will not need an entire page).

```

if self._items.is_empty():
    self._items.push(item)
else:
    temp = self._items.pop()
    self._enqueue(item)
    self._items.push(temp)

found = False
s2 = Stack
while not self._items.is_empty():
    temp = self._items.pop()
    if temp == item:
        found = True
        break
    self._items.push(temp)
    
```

[1 mark] Correctly moving all the items on to a temporary stack from self.\_items.

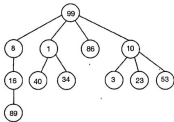
[1 mark] Uses 'found' to track if there was a match.



62390154-0804-4436-9A44-5511F930F010  
0ac148\_Midexam2-James3  
#630 Page 8 of 16

8. (6 pts) Trees

Consider the following Tree diagram:



a. (1 pt) Is this tree a Binary Search Tree (BST)? Clearly circle ONE option.  
Yes ☐ No ☒

b. (1 pt) What is the height of this tree?

c. (1 pt) Identify and write ALL the leaves present in the tree.

d. (1 pt) Clearly write the output when preorder.visit is called on this tree.

e. (1 pt) Clearly write the output when postorder.visit is called on this tree.

f. (1 pt) Clearly write the output when inorder.visit is called on this tree.

89, 16, 8, 40, 34, 1, 99, 86, 3, 10, 23, 53

Q8a

1

Q8b

1

Q8c

1

Q8d

1

Q8e

0


Q8f

0

5105171-8750-4507-0132-71289C5C9D1

001148\_Pdfmark2-10003

#630 Page 9 of 16



Use this page for rough work. If you want work on this page to be marked, please indicate this clearly at the location of the original question.

9





EC05041-0280-4836-A32C-684ED081843  
 cm148\_m00000-20003  
 #630 Page 10 of 16

#### 9. (2 pts) List comprehension

Convert the following code into one line using list comprehension.

##### PROGRAM:

```
L = [1, 2, 3, 4, 5]
"""Convert the code below into one line using list
comprehension."""
x = []
for i in L:
    if i % 4 == 3:
        x.append(i**2)
    else:
        x.append(i)
```

TODO: Write the implementation using list comprehension below.

~~x = [i\*\*2 for i in L if i % 4 == 3 else i]~~

Q9

1

The "for" statement should not be before the "if/else" condition. The correct solution is: `[i**2 if i % 4 == 3 else i for i in L]`  
 The correct syntax of list comprehension is:  
`[f(x) if condition else g(x) for x in sequence]`  
 or: `[f(x) for x in sequence if condition]`

1

C191AC95-2003-4610-A624-732B0C82603  
cas148.pdf?access-token=  
#630 Page 11 of 16

10. (8 pts) Mutating Trees

The aid sheet has the docstring for class `Tree`, which is relevant to this question.

Recall that the *depth* of the root of a tree is **depth 0**.

Consider the `Tree` method `trim`, with the docstring below:

```
class Tree:
    def trim(self, d: int) -> None:
        """Remove all values in the tree that are at depth <d> or greater.
```

*Precondition:  $d \geq 0$ .*

*Important notes:*

1. Calling `trim` with  $d = 0$  always results in an empty tree.
2. Calling `trim` when  $d$  is greater than the tree's height (number of levels) does not change the tree at all.

...

a. [2 pts] Suppose we have a variable `t` that is a `Tree` instance representing the following tree:

Here are two calls to `t.trim` with our initial tree `t`, shown above, but with different values of `d`. Below each call, draw what `t` would look like after the call. If the tree would be empty, write "empty".

`t.trim(2)`

`t.trim(0)`  

empty


Q10a

2 Good job! 2

11

Q10b

6



10760871-6A8B-4158-BA30-1851A7031A0  
cs16f\_midterm2-2ans3  
#630 Page 12 of 16

b. [6 pts] In the space below, implement the trim method using recursion. You may NOT use any tree methods other than is\_empty, but you may access all Tree attributes. You may not define any helper methods here. We have given you some code to help you get started.

```
def trim(self, d: int) -> None:
    if self.is_empty():
        pass
    else:
        if d == 0:
            self._root = None
            self._subtrees = []
        else:
            # In this branch, we know that self is non-empty and d > 0.
            for subtree in self._subtrees:
                subtree.trim(d-1)
```

Great 6 work!

12

Q11

1.5

11. (2 pts) Binary Search Trees

Briefly name/describe 4 properties of a Binary Search Tree.

- has at most two children
- all the values at the left are less than or equal to the root
- all the values at the right are greater than or equal to the root
- left child has lesser or equal value of its parent and right child

12. (5 pts) Efficiency and Big O

Consider the following implementation of negative integers in python:

```
def getbaseval(n: int) -> int:
    if n < 3:
        return n
    else:
        a = 0
        b = 1
        c = 2
        for i in range(3, n+1):
            d = a + 2*b + c
            a = b
            b = c
            c = d
        return d
```

a) (1 pt) Write the time complexity of the above implementation as a function of the input n, in Big O notation. Briefly provide reasoning in 2-3 sentences about the Big O you selected.

Since the function is not recursive, and has only one for loop, the time complexity is linear, thus it's  $O(n)$ .

This point is repeated -0.5


Q12a

1

https://app.crowdmark.com/score/ddd652c4-5be0-41bb-bbe4-1d9d98823df7?print=true

12/16

Q12b



22F93226-689D-48FF-88D5-8D4453F1F234  
cs014f\_midterm0-2ans3  
#630 Page 14 of 16

3

b) (3 pts) Write a Python program in the space below. Note: This program MUST implement a recursive algorithm for the above code. (You may NOT use memoization or dynamic programming techniques since it is out of scope for CSC148).

```
def gribonacci(n: int) -> int:
    if n == 2:
        return 1
    else:
        return gribonacci(n-3) + gribonacci(n-2) * 2
```

Has correct base case

1

Has correct recursive step

2

Q12c

0


c) (1 pt) Write the complexity of the recursive function you wrote in part b in Big O notation. Briefly provide reasoning in 2-3 sentences about the Big O you selected.

Since the function has 3 recursive calls in it, the time complexity will be  $O(3^n)$ .

Incorrect, the function makes three recursive calls to itself, making it  $O(3^n)$

0

01447475-10P4-455D-01P2-01C0P0P0A2N  
000148\_p0100000-00003  
#630 Page 15 of 16



[This page is for scratch work and will not be graded unless specified.]

15

