

# CSCI046 Final (Sample)

## **Collaboration policy:**

You may not:

1. discuss the exam with any human other than Mike; this includes:
  - (a) asking your friend for clarification about what a problem is asking
  - (b) asking your friend if they've completed the exam

You may:

1. take as much time as needed
2. use any written notes / electronic resources you would like
3. ask Mike to clarify questions via email

Name:

---

**Problem 1.** (3 pts) Complete each equation below by adding the symbol  $O$  if  $f = O(g)$ ,  $\Omega$  if  $f = \Omega(g)$ , or  $\Theta$  if  $f = \Theta(g)$ . The first row is completed for you as an example.

You will lose one point for each incorrect answer.

$f(n)$		$g(n)$
1	=	$O(n)$
$1.1^n$	=	$\Omega \ n^2$
$1/n$	=	$O \ 1$
$\log_4 n$	=	$\Theta \ \log_3 n$
$2^n$	=	$\Omega \ n^2$
$\log n$	=	$O \ \sqrt{n}$
$\log(n^3)$	=	$\Theta \ \log n$
$\log(n!)$	=	$\Theta \ n \log n$
$n!$	=	$\Omega \ n^2$
$(\log n)^4$	=	$\Omega \ (\log n)^3$

**Problem 2.** (15 pts) For each question below, circle either True or False. Each correct answer will result in +1 point, each incorrect answer will result in -1 point, and each blank answer in 0 points.

1. **TRUE** False Any procedure with worst-case runtime  $O(n)$  is guaranteed to have best case runtime  $O(n^2)$ .
2. **TRUE** False Given an array of  $n$  integers randomly generated between 1 and 1000, there exists an algorithm to sort these integers in time  $O(n)$ .
3. True **FALSE** Python's built-in `sort` function uses Quicksort.
4. True **FALSE** Merge sort is guaranteed to run asymptotically faster than insertion sort for all input lists.
5. **TRUE** False Python's built-in `sort` function allocates  $\Omega(n)$  memory.
6. True **FALSE** Python's built-in `range` function allocates  $\Theta(n)$  memory.
7. **TRUE** False In python 3, the `__contains__` method of the `set` class is asymptotically faster than the `__contains__` method of the `list` class.
8. **TRUE** False The worst case runtime for inserting into an AVL Tree is better than the worst case runtime for inserting into a BST.
9. True **FALSE** Python dictionaries are internally implemented using an AVL tree written in the C programming language.
10. **TRUE** False The height of every BST is guaranteed to be  $\Omega(\log n)$ .
11. True **FALSE** There does not exist a binary tree that is both a valid heap and a valid AVL tree.
12. True **FALSE** In python 3, a `list` variable can be used as the key for a dictionary.
13. True **FALSE** A pre-order traversal of a `heap` will traverse the elements in sorted order.
14. True **FALSE** If you release an open source project using the GPL license, the GPL license will guarantee that no one can make money selling your code without your permission.
15. True **FALSE** If you find a github repo with no `LICENSE` file, you may legally use code from this repo in a BSD3-licensed project.

**Problem 3.** Consider the following two functions:

---

```
1 def print_container_1(xs):
2     for i in range(len(xs)):
3         print('xs[i]=' ,xs[i])
4
5 def print_container_2(xs):
6     for i,x in enumerate(xs):
7         print('i,x=' , (i,x))
```

---

Notice that both functions will work whether the **xs** parameter is a list or a deque. Whenever a function works for more than one type of input, we call it **polymorphic**.

1. (1 pt) Assume we have a **list** with **n** elements. What is the runtime of `print_container_1` on this list? Your answer must be tight and in big-O notation.

$O(n)$

2. (1 pt) Assume we have a **list** with **n** elements. What is the runtime of `print_container_2` on this list? Your answer must be tight and in big-O notation.

$O(n)$

3. (1 pt) Assume we have a **deque** with **n** elements. What is the runtime of `print_container_1` on this deque? Your answer must be tight and in big-O notation.

$O(n^2)$

4. (1 pt) Assume we have a **deque** with **n** elements. What is the runtime of `print_container_2` on this deque? Your answer must be tight and in big-O notation.

$O(n)$

**Problem 4.** We saw in class that binary search runs in logarithmic time. One possible idea for speeding up the computation is to perform trinary search (that is, split the list into three sections rather than two on each recursive call). The following code implements trinary search.

---

```
1 def trinary_search(xs, val):
2     left = 0
3     right = len(xs)
4     def go(left, right):
5         if right-left < 3:
6             return val in xs[left:right]
7         mid1 = left + (right-left)//3
8         mid2 = left + (right-left)//3*2
9         if val < xs[mid1]:
10            return go(left, mid1)
11        elif val < xs[mid2]:
12            return go(mid1, mid2)
13        else:
14            return go(mid2, right)
15    return go(left, right)
```

---

1. (2 pts) What is the recurrence relation describing the runtime of `trinary_search`? Assume that the input data structure is a list.

$$T(n) = T(n/3) + 1$$

2. (2 pts) Solve the recurrence relation you gave for part 1 using the master theorem, and report your answer using  $\Theta$  notation. (You can get full credit on this problem even if you gave the wrong answer on the previous problem.)

$\Theta(\log n)$

**Problem 5.** In the questions below, assume that all data structures from the homeworks are correctly implemented.

1. (2 pts) What is the worst-case runtime of the `avl_sorted` function defined below in terms of the length  $n$  of the input list? (Use  $\Theta$  notation.)

---

```
1 from containers import AVLTree
2
3 def avl_sorted(xs):
4     avl = AVLTree(xs)
5     return xs.to_list('inorder')
```

---

$\Theta(n \log n)$

2. (2 pts) What is the runtime of the code below in terms of  $n$ ? (Use  $\Theta$  notation.)

---

```
1 from containers import BST
2
3 bst = BST()
4 for i in range(n):
5     bst.insert(i)
6     for x in bst.to_list('inorder'):
7         print("x=", x)
8     bst.remove(i)
```

---

$\Theta(n)$

**NOTE:** The first problem above is asking for a *worst case* runtime, whereas the second problem is only asking for a runtime. This is because in the first problem, there are many possible lists that could be input into the function, and each list might have different runtimes. In the second problem, there is only one particular sequence of values that will get inserted. Since there are not multiple possibilities, the worst/best/average cases for this problem are all going to be the same sequence. Note that this does not mean that the worst/best/average cases for the BST itself are the same. The final exam will test your ability to understand all of these subtleties.

**Problem 6.** The previous problems total 30 points. You will have 10 more points worth of problems related to material from the last 2 weeks of class.