COMP 285: Practice Midterm Questions

The following are questions meant to help you practice, and cannot be submitted for a grade.

Important Notes

- *It is meant to give you a chance to do some practice questions after having reviewed the slides, quizzes, in-class exercises, homeworks, etc.*
- *It should give a rough sense of some ways questions might be posed, though there's no guarantee that the actual midterm will have the exact same format (at a minimum, one difference is that the actual midterm will show the point values associated with the questions).*
- *It should give a rough idea of the level of mastery expected generally, though more/less mastery may be expected for any given topic.*

Thanks for reading the notes above - the big picture thing is that I want to be sure you use this resource appropriately, while at the same time **do not neglect the many other more comprehensive resources**!

Asymptotic Analysis

1. O(n/100 + log(n) + 200) can be simplified to O(n). True or False?

2. $2x + x^2/2 = \Theta(x^2 + 2x + x\log(x))$. True or False?

3. $x + 20 = \Omega(999)$. True or False?

For questions 4 - 6, refer to the containsDuplicates pseudocode.

```
algorithm containsDuplicates
  input: size n vector of ints called vec
  output: true if vec contains duplicates, false otherwise

for i = 0...n-1
  for j = i + 1...n-1  // Notice we start at i + 1, not j
    if vec[i] == vec[j]
      return true
return false
```

4. What is the **best-case runtime** of containsDuplicates? Define n, provide a tight upper bound with Big-O, and justify your answer.

5. What is the **worst-case runtime** of containsDuplicates? Define n, provide a tight upper bound with Big-O, and justify your answer.

6. What is the **worst-case space complexity** of containsDuplicates? Define n, provide a tight upper bound with Big-O, and justify your answer.

Using the Right Tools

7. Which of the data structure implementations below have O(1) runtime on average for element insertions? Select **ALL** that apply.
    - A stack (C++: std::stack)
    - A queue (C++: std::queue)
    - A hash set (C++: std::unordered_set)
    - A hash map (C++: std::unordered_map)
    - A vector (C++: std::vector)

8. Which of the data structure implementations below have O(1) runtime on average for element searching? Select **ALL** that apply.
    - A stack (C++: std::stack)
    - A queue (C++: std::queue)
    - A hash set (C++: std::unordered_set)
    - A hash map (C++: std::unordered_map)
    - A vector (C++: std::vector)

9. Which of the data structure implementations below have O(1) runtime on average for element removal? Select **ALL** that apply.
    - A stack (C++: std::stack)
    - A queue (C++: std::queue)
    - A hash set (C++: std::unordered_set)
    - A hash map (C++: std::unordered_map)
    - A vector (C++: std::vector)

Sorting

10. Which array of the following will CountingSort take the most number of steps on? Select **ONE.**
    a. [1, 2, 3, 4, 5, 6]
    b. [5, 43, 3, 11, 6, 9]
    c. [3, 1, 34, 3, 4, 81]
    d. [4, 4754, 4, 24, 1, 33]

11. For each of the below, explain in 1 - 2 sentences what they mean with respect to sorting.
    - Adaptive
    - Stability
    - In-Place

12. Given an array is already sorted, which sort will take the least time? Select **ONE.**
    a. Insertion Sort
    b. Quick Sort
    c. Merge Sort
    d. Selection Sort

For questions 12 - 13, refer to quickSort provided.

```
algorithm quickSort
  Input: vector<int> vec of size N
  Output: vector<int> with sorted elements

  if N < 2
    return vec
  pivot = findPivot(vec)
  left = new empty vec
  right = new empty vec
  for index i = 0, 1, 2, ... N-2
    if vec[i] <= pivot
      left.push_back(vec[i])
    else
      right.push_back(vec[i])
  return quickSort(left) + [pivot] + quickSort(right)
```

13. Suppose findPivot is a function which finds the element that will partition the list in two (nearly) equal halves in linear time while using constant space. What is the **worst-case runtime** of quickSort in this case? Justify your answer.

14. Challenge: what is the **worst-case space complexity** of quickSort in this case? Justify your answer.

Master Theorem

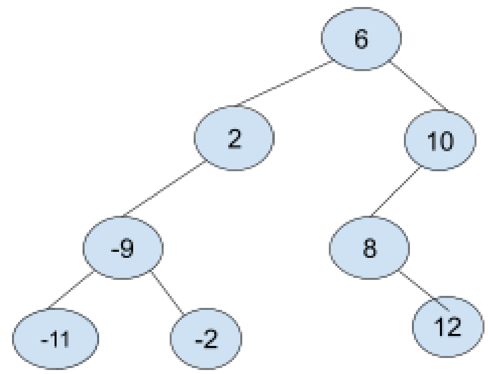
15. Find the runtime of an algorithm described by the following recurrence relation: $T(n) = T(n/2) + O(1)$. You may show your work for partial credit.

16. Write a recurrence relation for MergeSort. You may show your work for partial credit.

Trees

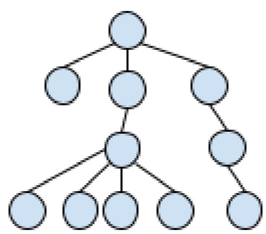
17. Is the tree on the right a Binary Search Tree? Explain.

18. What would an post-order traversal of this tree print out?



---

19. Complete the recursive case of countAtLevel in the box, which counts the number of nodes at each level in a Tree.



k = 0, # nodes = 1

k = 1, # nodes = 3

k = 2, # nodes = 2

k = 3, # nodes = 5

```
algorithm countAtLevel
  input: TreeNode root and a level k
  output: the number of nodes at level k in root

if level == 0  // base case
  return 1

total = 0
for each element child in root->getChildren()
   total += 
return total
```

20. What is the **best-case runtime** of searching for a node in a balanced BST. Give an example of when the best-case happens.

21. What is the **worst-case runtime** of **searching** for a node in a **balanced** BST. Give an example of when the worst-case happens.

For question 21, use the following pseudocode

```
BSTremove(t, v) // from visualgo.net
  search for v
  if v is a leaf
    delete leaf v
  else if v has 1 child
    bypass v
  else replace v with successor
```

22. Draw what the BST t below will look like after BSTremove(t, 8). (*Hint: Which node must replace the root node in order to maintain the BST property?*)