I&C SCI 46- Practice-MT1-S22 Solutions

Please note that this is intended as a guide for how to evaluate your own solutions and thought process; solutions posted, in general, are not "if you wrote exactly this, you'd get full marks."

1. (0.75 points) For each of the following functions, determine whether it is $\mathcal{O}(n)$, $\Omega(n)$, or both. You do not need to provide proof or justification. Make sure you clearly indicate, for each one, whether you believe it is $\mathcal{O}(n)$, $\Omega(n)$, or both. If the graders cannot determine your answer, you will not get credit for it.
   Note that we are *not* asking you to provide the "best" $\mathcal{O}$-notation/Ω-notation; merely to describe it with one or both of the choices provided.

$a(n) = n(\log n)^{10}$	☐ $\mathcal{O}(n)$	☐ $\Omega(n)$
$b(n) = n^{0.99}$	☐ $\mathcal{O}(n)$	☐ $\Omega(n)$
$c(n) = 10^{100}n$	☐ $\mathcal{O}(n)$	☐ $\Omega(n)$

   $a(n)$ is $\Omega(n)$ but not $\mathcal{O}(n)$. $b(n)$ is $\mathcal{O}(n)$ but not $\Omega(n)$. $c(n)$ is both.

2. (1.25 points) Rank the following functions in order from smallest asymptotic running time to largest. Additionally, identify a pair of functions x, y where $x(n) = \Theta(y(n))$. You do not need to show your work.

   For the ordering requirement, write the function identifier letters in order. Your submission should be exactly five characters. For example, if you believe the functions are currently listed in asymptotic order, your answer should be "abcde" (without the quotes).

   (a) $a(n) = 8^{(\log n)+1}$
   (b) $b(n) = n^8 \log n$
   (c) $c(n) = n \cdot (\log n)^8$
   (d) $d(n) = 10^4!$
   (e) $e(n) = 2^{6 \log \sqrt{n}}$

   $f_d < f_c < f_a = f_e < f_b$

   f_a and f_e are Θ of one another.

   note:

   $f_e = 2^{6 \log \sqrt{n}} = 2^{3 \log n} == (2^{\log n})^3 = n^3$

   $f_a = 8^{\log n+1} = 8 * 8^{\log n} = 8 * 2^{3 \log n} = 8 * (2^{\log n})^3 = 8n^3$

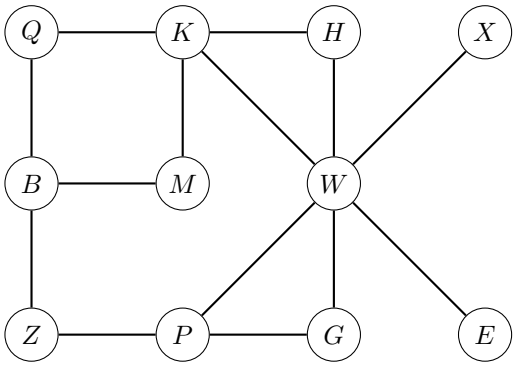
   (recall that all logarithms in this class are base two)

3. (4 points) Suppose I want to have a "Blue and Gold Stack" class. This is a class that has color-coded push, pop, top and size functions; for example, there is a "blue push" and a "gold push." Explain how you can implement this using a single array whose capacity is set at some value C, which will always be larger than the combined sizes of the blue and gold stacks. Your approach may only use the array of size C and $\mathcal{O}(1)$ additional space. You do not need to write code, but instead describe how you would implement the eight functions.

   You will earn no credit if you solve this by simply having two traditional stacks as private data members and redirecting the calls to `bluePush()` to the blue one, etc.

   This is similar to the array-based queue. Have a starting index for the blue stack and a starting index for the gold stack, as well as an index for the top. The blue stack pushes to smaller indices (wrap around, as the queue did) from its top and the gold stack pushes to large numbers. Pop and top work similarly, knowing where the top element is (and pop moving towards the start). Tracking size can be done by a private member variable that is incremented/decremented within push and pop as appropriate.

4. (2 point) Suppose we run a valid breadth-first order of the following graph, starting at vertex G. This means that vertex G is in layer zero on its own.



   Write the contents of each layer.

Layer	Contents
0	G
1	W P E
2	Z K H X
3	B M Q

5. (3 points) **The most disgusting recursive problem ever.**

   The McDonald's near campus sells Chicken McNuggets in orders of 6, 9, or 20. Suppose you are ordering for a party and you know exactly how many McNuggets will be eaten by guests. It turns out that, for any integer $n \geq 44$, you can order exactly n Chicken McNuggets at this McDonald's.

   For purposes of this problem, you cannot throw out McNuggets or allow them to go uneaten, such as by acquiring $n = 44$ by buying two twenty packs and a six pack, then discarding two. If the thought of this many Chicken McNuggets is too disgusting, you may pretend you are buying $n \geq 44$ celery sticks in bunches of 6, 9, or 20 (feel free to rename the function below in that case).

   Finish the recursive function below to complete the ordering and return the counts by reference parameters. You may assume for this problem that there will be no overflow or underflow at any point in the problem and that stack space is not a concern. The code has been started for you and is part of a correct solution.

```
void buyChicken (unsigned n, unsigned & num6Packs, unsigned & num9Packs,
                 unsigned & num20Packs)
{
  if( 44 == n ){
    num20Packs = 1;  num6Packs = 4;  num9Packs = 0;
  }else if (45 == n ){
    num20Packs = 0; num6Packs = 3;  num9Packs = 3;
  }else if (46 == n ){
    num20Packs = 2; num6Packs = 1; num9Packs = 0;
  }
```

   To answer this, fill in three more cases (for 47 through 49) and then have an else (for cases $n \geq 50$) that calls buyChicken for $n - 6$ and, after that returns, increments num6Packs once.

6. (2 points) Suppose I am writing a program that will use a skip list with maximum height $h = 3\lceil \log n \rceil$ - that is, when I add an element to the skip list, if I get $3\lceil \log n \rceil$ "heads" in a row, I will stop adding height to the element. You cannot modify my code or select the input. You can, however, change the way the coin works from "50% heads, 50% tails" (current status) to another set of probabilities (which must still add up to 100%). You cannot introduce other results: the coin can still only land on heads or tails (no sideways landing, for instance).
   Your goal is to cause my Skip List to use the **maximum possible amount of space**. How would you bias the coin to achieve this effect? Explain briefly why this will cause the given problem.

   Convert it to 100% heads. This will cause it to fill each newly added element to the top, each of which takes some space.

7. (4 points) Consider the following definition for a node in a linked list:

```
struct Node
{
  Node(unsigned v, Node * n) : value(v), next(n) {}
  unsigned value;
  Node * next;
};
```

Suppose we have a linked list of these and wish to answer a question: is there a subset of these Nodes such that the combined value equals a particular value 'target'?

Complete the following function that returns a bool value to indicate the answer to the question. You may assume that the initial call gives a link to the front of the linked list and that no overflow or underflow will occur at any point in the run of a program.

Hint: if there is a subset of the Nodes whose combined value adds to the target, then either the first element is in that subset or it isn't.

This problem is substantially easier if you use recursion. For many students, this will be the hardest question on the test. You may wish to consider answering it last.

```
bool subsetSum(Node * front, unsigned target)
{
```

Your code will look like this idea:

```
if target is 0 then
  return true
else if front is nullptr then
  return false
else if front→value > target then
  return subsetSum(front→next, target)
else
  return subsetSum(front→next, target) OR subsetSum(front→next, target - front→value)
end if
```