CMPT 306 Algorithms and Data Structures I
Midterm Examination
Fall 2017

100 Total Points

Selected Solutions

[1] 6 points) Solve (not big-Theta) the following recurrence relations.

- $C_n = C_{n-1} - 3$ where $C_0 = 1$ $\boxed{-3n + 1}$
- $C_n = 2\ C_{n-1} + 3$ where $C_0 = 3$ $\boxed{3 \times (2^{n+1} - 1)}$
- $C_n = -4\ C_{n-1}$ where $C_1 = 2$ $\boxed{-4^{n-1} \times 2}$

[2] (4 points) Determine the order of growth (using big-Theta) for the following recurrence relations. Where appropriate, you may use the master theorem.

- $C_n = 2\ C_{\frac{n}{2}} + n$ $\boxed{\Theta(n\ log\ n)}$
- $C_n = 4\ C_{\frac{n}{4}} + n^2$ $\boxed{\Theta(n^2)}$

[3] (5 points) An interesting type of tree is a *BRAID* tree which is defined as:

- $B_0 = 1$ (which means a BRAID tree of size 0 has one node);
- $B_N = 3\ B_{N-1} + 2$ (which means a BRAID tree of size N has $2 + 3 \times$ the number of nodes in a BRAID tree of size B_{N-1}).

Solve this recurrence relation to determine the number of nodes in a BRAID tree of size 20.

Solving this recurrence relation yields

$B_N = 2 \times 3^N - 1$

which generates $B_{20} = 2 \times 3^{20} - 1 = 6,973,568,801$

[4] (5 points) The Perseid meteor shower in August 2016 was especially brilliant! During the height of the shower, it was possible to determine the amount of time (in milliseconds) we could see a meteor by first determining the distance (in kilometers) from the Earth's surface to the meteor in the sky. A meteor K kilometers in the sky is visible 60% of the amount of time a meteor $K-1$ kilometers away + 10 milliseconds. All meteors are visible at least 10 milliseconds, so the base case is $K_0 = 10$. (Think of a meteor zero kilometers in the sky (i.e. on the ground) as being visible for 10 milliseconds.) This can be modeled with the recurrence relation: $K_N = .6\ K_{N-1} + 10$ where $K_0 = 10$.

How long (in milliseconds) is a meteor 100 kilometers in the sky visible for?

Solving this recurrence relation yields

$$K_N = 10 \times \left[\frac{.6^{N+1} - 1}{-.4}\right]$$

which yields $K_{100} = 10 \times \left[\frac{.6^{101} - 1}{-.4}\right] \approx 25\text{ms.}$

[1]

[5] (5 points) Consider the function `doodle()` shown in Figure 1.

```
def doodle(n):
    if n == 0:
        # draw point at random (x,y)
        drawPoint(random(), random())
    else:
        doodle(n-1)
        doodle(n-1)
```

Figure 1: `doodle()` function.

What is the recurrence relation that models this function? (You do not have to solve the recurrence, just provide it.)

$\boxed{C_n = 2\ C_{n-1} + 1}$

[1] You may find the Geometric series particularly helpful

$$\sum_{i=0}^{N} A^i = (A^{N+1} - 1)/(A - 1)$$

.

[6] (10 total points) Consider the algorithm shown in Figure 2 that is passed a binary tree `T` and it returns a value.

- (2 points) What will `mystery()` return if it is passed the tree shown in Figure 3? $\boxed{5}$
- (3 points) What operation does `mystery()` perform?
  It returns the number of nodes that have 2 children.
- (5 points) Set up a recurrence relation for `mystery()` and then determine its cost using Big-Theta notation.
  $C_n = 2\, C_{\frac{n}{2}} + 1$ which using master theorem yields $\Theta(n)$.

```
def mystery(T):
  if T is null:
    return 0
  else:
    left = mystery(left subtree of T)
    right = mystery(right subtree of T)

    count = left + right
     if (left subtree of T != null) and (right subtree of T) != null:
       count += 1

  return count
```

Figure 2: `mystery()` function.

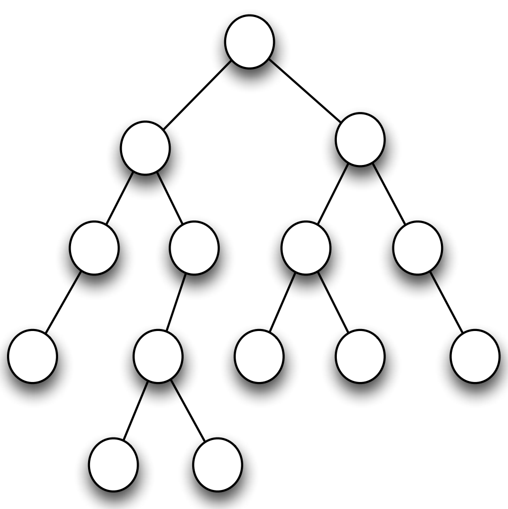

Figure 3: Example binary tree `T` that is passed to `mystery()` function.

[7] (10 points) A **Bentley Snowflake** is a 6-sided snowflake created from a set of n points in a Cartesian coordinate system. Bentley snowflakes occur when the air temperature is just above freezing (0 Celsius), and they can have very large branches (3-5 mm.) The snowflake is centered from a given (origin) point P and lines extend to 6 additional points that are closest to the origin.

- (6 points) Design an algorithm that is given a set of n points $\{ (x_1, y_1), \ldots (x_n, y_n) \}$ and an origin (x_i, y_i), draws a Bentley Snowflake.
- (4 points) What is the order of growth of your algorithm?

[8] (15 total points) Consider the *Leaf Counting* problem whereby you must design a recursive, divide-and-conquer algorithm for counting the number of leaves in a binary tree containing N nodes.

- (10 points) Write pseudocode for your algorithm.
- (5 points) Set up a recurrence relation for your algorithm and then determine its efficiency class using the Master Theorem.
  $C_n = 2\, C_{\frac{n}{2}} + 1$ which we apply the master theorem yields $\Theta(n)$.

[9] (15 total points) Consider the algorithm shown below for generating permutations.

Algorithm *HeapPermute*(n)
//Implements Heap's algorithm for generating permutations
//Input: A positive integer n and a global array $A[1..n]$
//Output: All permutations of elements of A
if $n = 1$
    write A
else
    for $i \leftarrow 1$ **to** n **do**
        HeapPermute($n - 1$)
        if n is odd
            swap $A[1]$ and $A[n]$
        else swap $A[i]$ and $A[n]$

This algorithm generates the permutations of the elements specified in the array `A` (where `A[1..n]`, not `A[0..n-1]`)

- (5 points) What is the time efficiency (in big-Theta notation) of this algorithm? (It may be worthwhile to trace the algorithm as it permutes the values of a, b, c)
  It generates all permutations of n elements, which is $n! = \Theta(n!)$ A recurrence relation is $C_n = n\, C_{n-1}$ where $C_1 = 1$.

- (10 points) Write a python implementation of this algorithm. Name your implementation `HeapPermute.py` and define a function named `permute(A,n)` where `A` is a list such as `A = ['a', 'b', 'c', 'd']` and `n` is the length of the list. (Don't ignore that Python list indices are zero-based – not one-based as the algorithm is presented.)

Don't confuse the name of this algorithm with the heap data structure; Heap *is the name of the designer of the algorithm.*