CMPT 306 Algorithms & Data Structures
Fall 2017
Homework #3
100 Total Points

Due Date: 11:59 PM Friday, October 13, 2017.
Submit your answers in a file through Canvas.

Be sure to show all supporting work to receive full credit.

SOLUTIONS

1. (30 points (6 pts each)) Solve (in closed form, not big-Theta) the following recurrences:

- $C_n = C_{n-1} + 5$ where $C_1 = 0$. Solution: $5(n-1)$
- $C_n = 3C_{n-1}$ where $C_1 = 4$. Solution: $3^{n-1} \cdot 4$
- $C_n = C_{n-1} + n$ where $C_0 = 0$. Solution: $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$
- $C_n = C_{(n/2)} + n$ where $C_1 = 1$. (Hint: use master theorem) Solution: $2 \cdot n - 1$
- $C_n = C_{(n/3)} + 1$ where $C_1 = 1$. (Hint: use master theorem) Solution: $1 + log_3\, n$

2. (4 points) Consider the following problem: Design an algorithm to determine the best route for a passenger to travel from one location to another using public transportation where travel may involve (a) bus, (b) light rail, (c) walking, and (d) commuter rail. For example, a passenger may have to ride a bus to one station, walk to another station where they get on light rail, then take a bus to to another bus, and so forth.

- What reasonable criteria could be used for defining the "best" route? Best route could really be based upon one (or more) of several criteria: (a) the fastest trip; (b) the trip with the fewest stops; (c) the trip with the fewest train changes, etc.
- How would you model this problem as a graph? As a graph, each station/stop represents a vertex, each edge is represented as a route from one station to another.

3. (6 points) Describe how one can implement each of the following operations on an array so that the time it takes does not depend on the array size n.

- Delete the i^{th} element of an unordered array of size n. Replace the i^{th} element with the last element and decrease the size of the array by 1.

- Delete the i^{th} element of an ordered array, where the array remains in sorted order after the deletion. Replace the i^{th} element with a special symbol (called a *sentinel*) that cannot exist in the data of the array. This sentinel element must be skipped over when encountered when processing the array.

4. (12 points) List the following expressions from best(lowest) to worst(highest) order. If any expressions are of the same order, indicate that they are equal. (Note lg refers to log_2)

$$2^n,\ n - n^2 + 5n^3,\ 2^{n-1},\ lg\ n,\ n^3,\ n\ lg\ n,\ n^2,\ \sqrt{n},\ 42,\ n,\ (3/2)^n,\ n!,\ n^3 + lg\ n$$

42
$lg\ n$
$\sqrt{n}$
n
$n\ lg\ n$
n^2
$n - n^2 + 5n^3,\ n^3,\ n - n^2 + 5n^3$
$(3/2)^n$
$2^n,\ 2^{n-1}$
$n!$

5. (8 points) Algorithms W, X, Y, and Z are analyzed and found to have worst-case running times no greater than $20 \times Nlog_{10}N$, $5 \times N^2$, $.005N^3$, and $5000 \times N$ respectively. Answer the following questions:

- (4 points) What is the big-theta notation of each of these four algorithms?
  $\Theta(N\ log\ N)$, $\Theta(N^2)$, $\Theta(N^3)$, $\Theta(N)$
- (4 points) Using your answer from the question above, what is the ordering of these four algorithms from best(lowest) to worst(highest)?
  $\Theta(N)$, $\Theta(N\ log\ N)$, $\Theta(N^2)$, $\Theta(N^3)$ or Z, W, X, Y.

6. (20 points) Consider the following Python code:

```
def mystery(n):
  s = 0

  for i in range(1,n+1):
    s = s + i * i

  return s
```

- (6 points) What does `mystery()` compute?

  This algorithm computes $\sum_{i=1}^{n} i^2$

- (2 point) What is its basic operation?

  The basic operation is the statement `s = s + i * i`

- (2 point) How many times is its basic operation executed?

  The basic operation is executed n times.

- (4 points) What is its efficiency using Big-Theta notation?

  $\Theta(n)$

- (6 points) Can you suggest an improvement to this algorithm? If so, what is the efficiency class of your improvement? Otherwise, try to provde that it cannot be done.

  One improvement would be to use the closed form notation

  $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

7. (20 points) Consider the following recursive algorithm for computing the sum of the first n cubes: $S(n) = 1^3 + 2^3 + ... + n^3$:

```
def sum(n):
  if n == 1:
    return 1
  else:
    return sum(n-1) + n * n * n
```

- (12 points) Set up and solve a recurrence relation for the number of times the algorithm's basic operation is performed.

  Solution: A recurrence relation would be $C_n = C_{n-1} + 2$ where $C_1 = 0$. The term C_{n-1} means we are recursively solving a problem one less than n, the $+2$ represents the 2 multiplications. (We really could put *any* constant number here.) Its solution is $2(n-1)$ which is $\theta(n)$.

- (8 points) How does this algorithm compare with the straightforward, nonrecursive algorithm for computing this sum?

  Solution: A non-recursive solution will have a for loop ranging from 2 to n, so it will still have the same linear $\theta(n)$ performance as the recursive solution. The only advantage of the iterative, non-recursive solution is that it avoids the overhead of the space requirements on the stack of making the n recursive calls.