COMPSCI 320SC 2019 Midterm Test

University ID: _____

Student Name: _____

Student Signature: _____

Time Finished: _____

Attempt *all* questions. (Use of calculators is NOT permitted.)

Put the answers in the space below the questions.

Write clearly and *show all your work*!

Marks for each question are shown just before each answer area.

This 60 minute test is worth 10% of your final grade for the course.

Good luck!

Question #:	1	2	3	Total
Possible marks:	10	10	10	30
Awarded marks:				

1. Basic Analysis **[10 marks]**

(a) Show that $\frac{\ln n}{\sqrt{n}}$ is $O(\sqrt{n})$. ***(3 marks)***

Using limit rule: $\lim_{n\to\infty} \frac{\ln n}{\sqrt{n}}/\sqrt{n} = \lim_{n\to\infty} \frac{\ln n}{n} = 0$

(b) Prove that if f_1 is $\Omega(g_1)$ and f_2 is $\Omega(g_2)$, then $f_1 + f_2$ is $\Omega(g_1 + g_2)$. ***(3 marks)***

There exists $c_1 > 0$ and $n_1 > 0$ such that $0 \leq f_1(n) \geq c_1 \cdot g_1(n)$ for all $n \geq n_1$.
There exists $c_2 > 0$ and $n_2 > 0$ such that $0 \leq f_2(n) \geq c_2 \cdot g_2(n)$ for all $n \geq n_2$.
Then, $f_1(n) + f_2(n) \geq \min(c_1, c_2)(g_1(n) + g_2(n))$ for all $n \geq \max(n_1, n_2)$.

(c) Show that if $f(n) = \left(\frac{n}{2}\right)^n$ then $f(n) = 2^{\Theta(n \log n)}$. ***(4 marks)***

Since $\log_2 f(n) = n \log_2 \frac{n}{2} = n(\log_2 n - 1) = \Theta(n \log n)$, we have

$$2^{\log_2 f(n)} = f(n) = 2^{\Theta(n \log n)}.$$

2. Divide and Conquer **[10 marks]**

(a) Master theorem: Consider the following "divide-and-conquer" function:

```
function printer(int n)
  for i = 1 to n do
    for j = i + 1 to n do
      print CS320
    end for
  end for
  if n > 0 then
    for i = 1 to 4 do
      printer(⌊n/2⌋)
    end for
  end if
```

Let $T(n)$ denote the number of CS320 generated by a call of $printer(n)$.

i. Provide a recurrence equation for $T(n)$. ***(3 marks)***

$$T(n) = \begin{cases} 0, & \text{if } n \leq 1 \\ 4T(\lfloor n/2 \rfloor) + n(n-1)/2, & \text{if } n > 1 \end{cases}$$

The second term can be replaced by $\Theta(n^2)$ or $O(n^2)$.

ii. Solve the recurrence asymptotically for general n. ***(2 marks)***

Apply the master recurrence theorem with $a = 4$, $b = 2$, $c = 2$, we have $T(n) = \Theta(n^2 \log n)$ or $O(n^2 \log n)$.

You may want to make use of the following *master recurrence theorem*: Assume $T(n) = aT(n/b) + g(n)$, where $g(n)$ is $O(n^c)$, is the total time for a divide and conquer algorithm. Then:

$$T(n) = \begin{cases} O(n^c), & \text{if } a < b^c \\ O(n^c \log n), & \text{if } a = b^c \\ O(n^{\log_b a}), & \text{if } a > b^c \end{cases}$$

(b) Finding the peak item in array

A peak item in an array is the item that is greater than its neighbors. If there are more than one peak item, simply return one of them.

Input: [1, 5, 3, 2, 4, 0]
Output: 4
Input: [1, 2, 3, 4, 5, 6]
Output: 6
Input: [7, 6, 5, 4, 3, 2]
Output: 7

Describe a divide-and-conquer algorithm that solves this problem in $O(\log n)$ time where n is the size of the array. (***5 marks***)

We simulate the binary search method by first retrieving the mid item.
If it is greater than both of its neighbors, return it (2 marks).
If the left (or right) neighbor is greater than the mid item, recursively find the peak on the left (or right) part of the array (2 marks).
In the worst-case data (example 2 and 3), we need $O(\log n)$ time to return the peak item (1 mark).
We note that if there is more than one peak element, this solution will return one of them.

3. Greedy Algorithms **[10 marks]**

(a) Show how the greedy algorithm for making change gives minimum number of coins for \$7.95 using only New Zealand coinage (200, 100, 50, 20, 10 and 5 cents). Explain your answer. . ***(3 marks)***

795 - 200 - 200 - 200 - 100 - 50 - 20 - 20 - 5 = 0 implies 8 coins

(b) List and briefly describe the three methods presented in class for proving correctness of greedy algorithms. ***(3 marks)***

1. *Greedy algorithm stays ahead*. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithms.
2. *Structural*. Discover a simple structural bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.
3. *Exchange argument.* Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

(c) Consider the following simple greedy algorithm for properly coloring the vertices of a graph. [Recall that a graph is *properly colored* if we can assign colors to vertices such that adjacent vertices have different colors.]

function *GreedyColor*(Graph $G = (V, E)$)
Sort V by their degree in non-increasing order (e.g. largest to smallest)
ColorsAvail = $\{1, 2, \ldots, |V|\}$; ColorsUsed = $\{\}$
for each $v \in V$ (preserving sorted order) **do**
    Let c be smallest positive integer in ColorsAvail not used for a neighbor of v
    Color[v] = c
    ColorsUsed = ColorsUsed $\cup \{c\}$
return Color, |ColorsUsed|

Show with a small counter-example that this greedy algorithm does not always give the minimum number of colors. (Justify your answer by showing a valid smaller proper coloring.) ***(4 marks)***

A simple counter-example is the following graph in adjacency lists format.
0: 2 6 7 (colored 1)
1: 3 4 5 (colored 1)
2: 0 3 (colored 2)
3: 1 2 (colored 3)
4: 1 (colored 2)
5: 1 (colored 2)
6: 0 (colored 2)
7: 0 (colored 2)
However, this is a tree (hence bipartite/2-colorable).