Analysis of Algorithms **Comprehensive Examination** **Spring 2018**

Sign this exam with your student number — not your name: _____

1. (10 pts) Find a simple expression for the summation:

$$\sum_{0 \le k < n} (n-k)k$$

2. (10 pts) Solve the recurrence $T(n) = 3T(n-1) + 1$ with initial condition $T(0) = 1$.

3. (5 pts) (`True` or `False`) Explain your answer. If $f(n) = O(\sqrt{n})$ then $f(n) = O(n)$.

4. (5 pts) (`True` or `False`) Explain your answer. $\sqrt{n} = O(\lg n)$.

5. Consider the code below that computes the edit distance between two strings.

```
editDistance :: (Eq a) => [a] -> [a] -> Int
editDistance xs [] = length xs    -- delete xs to match the empty string
editDistance [] ys = length ys    -- insert ys into the empty string
editDistance (x:xs) (y:ys)
   | x == y =  editDistance xs ys
   | otherwise = minimum [ 1 + editDistance xs (y:ys),
                           1 + editDistance (x:xs) ys,
                           2 + editDistance xs ys]
```

(a) (10 pts) Assume length (x:xs) + length (y:ys) = n. What equation models the time complexity of the otherwise case?

(b) (10 pts) Classify the time complexity as either polynomial or exponential. Give reasons for your classification.

6. (10 pts)
The code below also computes the edit distance between two strings. What is its time complexity and what is the fundamental difference between this implementation and the code in the previous problem?

```
#include <stdio.h>
#include <string.h>

int editDistance(const char *s, const char *t) {
  int n = strlen(s), m = strlen(t);
  int d[n + 1][m + 1];

  for (int i = 0; i <= n; i++) { d[i][0] = i; }
  for (int j = 0; j <= m; j++) { d[0][j] = j; }

  for (int i = 1; i <= n; i++) {
     for (int j = 1; j <=m; j++) {
        int a = d(i-1,j) + 1;
        int b = d(i-1,j) + 1;
        int c = d(i-1, j-1) + (s[i] != t[j]);
        d[i, j] = min(a, b, c);
     }
  }
  return d(n, m);
}
```

7. The knapsack problem can be framed as a container that can hold a weight capacity C together with a list of provisions p_i, $i = 0, \ldots, n$ to be placed in the container. Each provision has a corresponding weight w_i and value v_i. The provisions are to be placed into the container without exceeding the capacity C and such that the sum of values is maximized.

   (a) (5 pts) Describe a couple of greedy heuristic that could be used to fill the container.

   (b) (5 pts) Assume that fractional amounts $0 \leq f \leq 1$ of each provision can be used. Describe an optimal filling strategy and its time complexity.

   (c) (10 pts) Now assume an all or nothing (0 or 1) approach must be used. Describe an algorithmic approach, and its complexity, that could be used in this case.

Brief Answer Questions

(20 pts)

1. Depth-first traversal on a graph can produce a spanning tree.

2. Breadth-first traversal on a graph can produce a spanning tree.

3. Neither depth-first not breadth-first traversal on a graph can produce a spanning tree.

4. If the time complexity of an algorithm is $O(f(n))$ what can be inferred about its space complexity?

5. Consider the complexity classes: P, NP, NP-complete and NP-hard
   (a) `True` or `False`: $P \subseteq NP$.

   (b) `True` or `False`: NP-complete $\subseteq NP$.

   (c) `True` or `False`: NP-hard $\subseteq NP$-complete.

   (d) `True` or `False`: If there is a reduction (deterministic polynomial time transformation) from every problem $X \in NP$ to Y, then Y is NP-complete.

   (e) `True` or `False`: One way to prove a problem $X \in NP$-hard is to develop a reduction from a known NP-hard problem Y to X.

6. Suppose, there exists a reduction (a polynomial time transformation) of instances of problem Y to instances of problem X.
   (a) If $X \in P$ what can you conclude about the complexity of Y?

   (b) If Y is undecidable, what can you conclude about the complexity of X?

7. `True` or `False`: Non-deterministic algorithms can be simulated with deterministic algorithms.