Name:

CSE 4081 Introduction to Analysis of Algorithms

Fall 2017 Final

Score

1. (10 pts) Use summation notation to show the pseudo-code below has time complexity $\Theta(n^3)$.

```
int s = 0;
for (int i = 0; i < n; i++) {
  for (int j = 0; j < i; j++) {
    for (int k = 0; k < j; k++) { s = s + 1; }
  }
}
```

For extra consideration, compile and run the code printing the value of s to see if it agrees with your analysis.

Score

2. (10 pts) Let $f, g : \mathbb{N} \mapsto \mathbb{R}$ be functions on the naturals to the reals. What does it mean to say $f(n) = O(g(n))$? A precise definition and picture could be a good answer.

Score

3. (10 pts) Assume $f(n) = O(g(n))$. Carefully show that $f(n)^2 = O(g(n)^2)$.

Score

4. (10 pts) It is commonly said that polynomial algorithms are tractable, but exponential algorithms are not. Explain this comment by demonstrating the increase in time when the input size is doubled for the functions below.

   (a) $f(n) = n^3$

   (b) $f(n) = 2^n$

Score

5. (10 pts) Show that $n + n \log_3 n$ solves the recurrence

$$T_n = 3T(n/3) + n, \quad T(1) = 1$$

Score

6. (10 pts) Show that you can solve the recurrence $T_n = 3T(n/3) + n$ with $T(1) = 1$ by some method other than guess-work.

Score

7. (10 pts) Show how to solve the recurrence $T(n) = T(n-1) + 2^{n-1}$ with $T(0) = 0$ using generating functions concepts.

Score

8. (15 pts) Consider the problem of scheduling chores: Pretend n chores c_k, $k = 0, \ldots, (n-1)$ each of which takes time t_k, $k = 0, \ldots, (n-1)$ and has profit p_k, $k = 0, \ldots, (n-1)$ are to be scheduled within a time M. Also, chores must be scheduled sequentially, but no other order is enforced.

The objective is to schedule the chores so that the maximum profit is achieved within the time constraint. Your task is to compare divide-and-conquer, dynamic programming, and greedy algorithms for this problem.

(a) The divide-and-conquer algorithm below purports to find the maximum profit of a set of n chores within the time bound M.

```
int maxProfit(int n, int M) {
  if (t[n-1] > M) {
    return maxProfit(n-1, M);
  }
  else { return max( maxProfit(n-1, M), p[n-1] + maxProfit(n-1, M-t[n-1])); }
}
```

What is the is the worst case time complexity of maxProfit?

(b) And alternative approach is to use the same recurrence but evaluate it bottom-up saving results for later re-use.

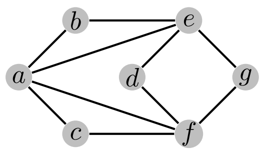
```
int maxProfit2(int n, int M) {
  int K[n][M];
  for (int i = 0; i < n; i++) {
    for (t = 0; t <= M t++) {
      if (i==0 || t==0) { K[i][t] = 0; }
      else if (t[i-1] <= M) {
        K[i][t] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
      }
      else { K[i][w] = K[i-1][w]; }
    }
  }
  return K[n-1][M-1];
}
```

What is the worst case time complexity of maxProfit2?

(c) Describe a greedy algorithm maxProfit3 for the problem. What is its time complexity? Does it always produce an optimal solution?

Score

9. (15 pts) Let G=(V,E) be a graph with n nodes and e edges. A subset S of vertices is independent when there are no edges between pairs in S. For instance $\{b, d, c, g\}$ is an independent set in the graph below



   (a) How many subsets $\mathbb{S}$ of size k does an n element set have?

   (b) If $\mathbb{S}$ is a subset of k nodes, how many *potential* edges (u, v), $u \neq v$ are there?

   (c) The algorithm below tests if graph G has a k-element independent set of size k.

```
bool independentSet(graph G, int k) {
  for (each k-element subset S of V)  {
    if (no pair (u, v) of (different) nodes in S is an edge in E) {
      return true;
    }
  }
  return false;
```

   Pretend the time complexity is controlled only by the for loop and number of tests in the **if** statement. With these assumptions and your answers to problems 9a and 9b, what is the worst case time complexity of the code?

   (d) Consider the algorithm below that finds maximum size independent sets within a graph G. What is its worst case time complexity?

```
int maxIndependentSet(Graph G {
  int max = 1;
  for (int k = 2; k <= n; k++) {
    if (independentSet(G, k) && (k > max)) { max = k; }
  }
  return max;
```

   (e) Show that there is a polynomial time algorithm that when given a certificate independent set of vertices checks that it is indeed independent. Conclude that IndependentSet is in NP.