COMP 7712 - Fall 2016
Exam 3 Name: _____

ONLY FRONT PAGES ARE DIGITIZED AND GRADED.

1. (*10 points*) Find the running time complexity of the following Python program. Explain your answer.

```
# L is a list of numbers
def f(L):
    if len(L) < 1:
        return 1
    A = L[0: len(L)//2]
    s = 0
    for x in A:
        for y in S:
           s = x + y
    return s * f(A)
```

2. (*10 points*) Find the space complexity of the Python program **f** in the previous question. Explain your answer.

3. (*20 points*) Write a Python program that takes as input a list L and prints out all intervals of L. An interval starts with an index i and ends at an index j ($j \geq i$). For example, $L = [1, 3, 5, 7, 9]$, then one interval is 3,5,7. You program must print out all intervals.

4. (*20 points*) Use back tracking to write a Python program that takes as input a list L and prints out all subsequences of L. For example, $L = [1, 3, 5, 7, 9]$, then one subsequence is 3,9; another subsequence is 1,5,7. You program must print out all subsequences. Hint: represent subsequences as **sets**. Config[i] = True if L[i] is part of the subsequence.

5. (*10 points*) Explain why this function foo might repeatedly calculate the same computations.

```
def foo(a, b, flag):
    if a < 0 or b < 0:
        return a+b
    if flag:
        return foo(a-1, b) + foo(a, b-1) + foo(a-1, b-1)
    else:
        return foo(a-1, b-1)
```

6. (*20 points*) Rewrite this function to do the same calculations, but use a cache Table to store computations to faster computation.

7. (*10 points*) Consider a variant of the Knapsack problem, in which each item can be taken more than one times (image you have an infinite amount for each item). The inputs are Weights, Calories and Capacity. The output is the maximum calories that can be taken to fit the bag with given Capacity. For example, with Weights = [6,3,4,2], Calories = [3000,1700,1600,900], and Capacity 10; then the best solution gives maximum calories 5200 (1700+1700+900+900).

```
def Knapsack(Weights, Calories, Capacity):
    if Capacity <= 0:
        return 0

    # Hint: imagine you take items sequentially and place into the optimal solution.
    # The question is which item should you take next.  If the next item is i, then
    # what is the resulting subproblem?  Although you do not know what i is, you
    # know that (1) i must be between 0 and len(Weights)-1 and (2) Weights[i] <= Capacity.
```

8. (*10 points*) A graph G has n vertices and m edges. Each edge (u, v) that connects two vertices u and v has a distance $G[u, v]$. $G.to(v)$ returns a list of vertices pointing to v. $G.from(v)$ returns a list of vertices that v points to.

Assume that G has no cycle. Given a starting vertex s, complete the following program to find the shortest distance from s to v.

```
def shortest_distance(G, s, v):
    if v == s:
        return 0
    if G.to(v) == []:
        return Infinity   # assuming Infinity is a very large number
    else:
        # Hint: the shortest path from s to v must go through a vertex u before getting to v
        # Visually, it's like this: s -> .... -> u -> v
        # We don't know what u is, but we know u must be in G.to(v)
```