Homework 8

Due November 20, 2019

**(a)** The Mandrill and Baboon are two *fictitious* problems that accept a weight graph and return an integer. Consider the reduction below, which reduces the Mandrill problem to the Baboon problem.

```
Input: G = (V, E): weighted graph with n vertices and m edges
Input: n, m: order and size of G
Output: Mandrill(G)
Algorithm: MonkeyBusiness
H = Graph(n);
for u ∈ V do
    for v ∈ V do
        if G.isAdjacent(u, v) then
            Let w be the weight of (u, v);
            if w > 100 then
                H.addEdge(u, v, 5 + w/20);
            else if w > 25 then
                H.addEdge(u, v, √w);
            end
        end
    end
end
return Baboon(H);
```

1. What is the worst-case time complexity of this reduction if we use an adjacency matrix to represent the graphs G and H?

**MonkeyBusiness = O(n$^2$) + O(Baboon(H))**

2. Suppose that we know that the complexity for Baboon is bounded above $O(B(m,n))$ and below by $\Omega(b(m,n))$ for a graph with *m* edges and *n* vertices. What does the algorithm above prove about the complexity of Mandrill, assuming that *B(m, n)* and *b(m,n)* are both larger than your answer to question 1? Justify your answer.

$$\begin{tikzcd} O(B(m,n)) & & O(B(m,n)) \arrow[ll] \\ \text{Mandrill} & \leq_p & \text{Baboon} \\ ?? \arrow[rr] & & \Theta(B(m,n)) \end{tikzcd}$$

**Mandrill = O(B(m,n)). We cannot determine with the information given the lower bounds of Mandrill.**

**(b)** The Knapsack Problem *(KP)* gives you a target integer *t* and an array *data* with *n* positive integers, and it returns the subset of *data* with a sum as close as possible to *t* without going over.

The Knapsack Size Problem *(KSP)* gives you a target integer *t* and an array *data* with *n* positive integers, and it returns the largest integer less than *t* that some subset of *data* sums up to.

The Full Knapsack Problem *(FKP)* gives you a target integer *t* and an array *data* with *n* positive integers, and it returns **true** if there is a subset of *data* that sums to exactly *t* and **false** otherwise.

1. Prove that $FKP \in NP$ by giving pseudocode for polynomial-time verification algorithm for *FKP*.

```
|   sum = 0
|   for i in range(len(subset):
|           if subset[i] is not in data
|                   return false
|           sum+= subset[i]
|   if sum == t
|           return true
|   else
|           return false
```

2. What kind of reduction do you need between *FKP* and *KSP* to prove the complexity of *KSP* is polynomial if $FKP \in P$? Justify your answer.

**To prove *KSP* is polynomial, we must go from our unknown, KSP to our Known FKP**

$$\begin{array}{ccc} p & \longleftarrow & p \\ \text{KSP} & \le_p & \text{FKP} \end{array}$$