CS 5592: Design and Analysis of Algorithms
Homework 5
Author: Hayden McParlane

(1.a) Prove that this decision problem is in the NP-class.

Proof that a decision problem is in the NP-class simply requires writing a verification algorithm that can verify a given certificate in polynomial time.

Algorithm 1 A verification algorithm for the problem above. This algorithm assumes that the projects in need of completion, P, are known to keep its format consistent with typical NP-class verification algorithms.

```
1: function VERIFY(C, k)
2:     for project in P do
3:         for company in k do
4:             1. Retrieve projects from L completable by the current company.
5:             2. Remove the project from the project list.
6:         end for
7:     end for
8:     if project list is empty then
9:         return Yes
10:    else
11:        return No
12:    end if
13: end function
```

This algorithm proves that the problem is in the NP-class because it executes the verification in polynomial time. For each project chosen, all of the companies are checked to determine which projects that company can complete. If a company is found that can complete every project, the algorithm returns yes. Otherwise, no is returned. This is polynomially related to the number of projects in P because the problem states that all of the projects can be completed but that none of the companies can complete all of the projects alone. This means that the upper bound on the list of projects associated with a given company is $|P| = m$. So, even if each company could complete all of the projects the verification have the upper bound $|P| \cdot (!|P|)^k \leq |P| \cdot |P|^k = |P|^{k+1} = m^{k+1} = m^{constant}$. Therefore, this problem is no harder than the NP-class and belongs to the NP-class.

(1.c) Suppose you have an algorithm A(P, m, C, n, k) that can determine, for any given integer k, whether k companies are enough to handle all projects. (It gives a "yes" or "no" answer.). Please show how we can use the algorithm A(P, m, C, n, k) to select the smallest number k of companies to finish all m projects so we can maximally save our budget. You need to output the companies that have been selected by your method (algorithm).

Algorithm 2 An algorithm that uses the given function to find the minimum cost. This algorithm assumes that the cost of each company is the same.

```
1: function FindMinCost(P, m, C, n)
2:     k ← n
3:     repeat
4:         k ← k − 1
5:     until A(P, m, C, n, k) equals NO
6:     k ← k + 1
7:     while sizeOf(C) > 0 do
8:         current ← pop(C)
9:         if A(P, n, C, n, k) equals NO then
10:            C ← current ⋃ C
11:            break from while
12:        end if
13:    end while
14:    return C
15: end function
```

This algorithm starts off by giving k a value of n. The problem statement states that n companies will be able to complete all of the projects so we know the result of a call to A will be YES giving us the ability to skip that call. Each iteration of the repeat decrements k until the call to A returns No at which point the loop breaks. When this occurs, the current value of k received a No so we can assume that the smallest k would be the previous k value; k + 1. Therefore, k is assigned k + 1.

The while-loop following that assignment ensures that companies are still present in C. If there are companies present, a next company is popped off of C at which point another call to A is done to determine if removal of that company from C resulted in No. During the while portion, k is kept constant while the elements of C vary. When the call to A is done and results in no in the while-loop, we can assume that the previously eliminated company was vital and needs to be in the solution.

(2) Consider the following activity selection problem. Suppose we have two lecture halls which are available from time t = 0. There are n activities, $a_1, a_2, a_3, ..., a_n$, that apply for using either one of these two lecture halls. Each activity a_i, $(1 \leq i \leq n)$ has flexible starting time but must start before or at s_i (≥ 0) and requests to use the hall for t_i hours. We wish to select the largest set of scheduled activities such that their requests can be satisfied with no time conflict. That is, they can be scheduled into one of these two lecture halls such that, at any time, at most one activity takes place in each hall.

A Re-define this problem as a decision problem.

B Prove that this decision problem is NP-hard.

2.A Re-defining this problem as a decision problem can be done as follows.

Consider the following activity selection problem. Suppose we have two lecture halls which are available from time t = 0. There are n activities, $a_1, a_2, a_3, ..., a_n$, that apply for using either one of these two lecture halls. Each activity a_i, $(1 \leq i \leq n)$ has flexible starting time but must start before or at s_i (≥ 0) and requests to use the hall for t_i hours. **Can we find a set of k activities such that all k activities can be scheduled in the two lecture halls without time conflict?**

2.B A fairly intuitive problem to map to our new problem above in this proof is the set partion problem.

The set partition problem is defined as follows. Given a set of numbers, $S = \{a_1, a_2, a_3, ..., a_n\}$, can we partition S into two sets, A and B, such that the following is true:

$$\sum_{i \in A} v_i = \sum_{i \in B} v_i$$

It's important to realize that in order for the above to be true, the total sum of every element in the set S must be M meaning that each partition must sum to $M/2$. That is,

$$\sum_{i \in S} v_i = M$$

We further assume that all elements of the set S have a value less than M/2. If this weren't the case partitioning the set wouldn't be possible which is an extraneous case not in need of consideration here. With this in mind, we can map the set partition problem to that above in the following manner:

1. Take each set value, v_i, and map it to a corresponding duration t_i.
2. Take each set value, v_i, and map it to a latest starting time s_i using the following equation:

   $s_i = (M/2) - t_i$

3. Add two new values, a_{n+1} and a_{n+2} such that

$$a_{n+1} =$$