CS41 Homework 10

This is a two-week, **14 point assignment**. This homework is due at 11:59PM on Sunday, December 8. Write your solution using LaTeX. Submit this homework in a file named `hw10.tex` using **github**. This is a partnered homework. You should primarily be discussing this homework with your partner.

It's ok to discuss approaches at a high level. In fact, I encourage you to discuss general strategies. However, you should not reveal specific details of a solution, nor should you show your written solution to anyone else. The only exception to this rule is work you've done with a lab partner/group *while in lab*. In this case, note (in your post-homework survey) who you've worked with and what parts were solved during lab.

The main **learning goals** of this homework are to practice working with NP-Completeness and to design and analyze approximation algorithms.

1. Multiple-Interval-Scheduling (K&T 8.14) In this problem, there is a machine that is available to run jobs over some period of time, say 9AM to 5PM.

   People submit jobs to run on the processor; the processor can only work on one job at any single point in time. However, in this problem, each job requires a **set of intervals** of time during which it needs to use the machine. Thus, for example, one job could require the processor from 10AM to 11AM and again from 2PM to 3PM. If you accept this job, it ties up your machine during these two hours, but you could still accept jobs that need any other time periods (including the hours from 11AM to 2PM).

   Now, you're given an integer k and a set of n jobs, each specified by a set of time intervals, and you want to answer the following question: is it possible to accept at least k of the jobs so that no two of the accepted jobs have any overlap in time?

   In this problem, you are to show that Multiple-Interval-Scheduling $\in$ NP-complete. To assist you, we've broken down this problem into smaller parts:

   (a) First, show that Multiple-Interval-Scheduling $\in$ NP.

   (b) In the remaining two parts, you will reduce

   $$\text{Independent-Set} \leqslant_P \text{Multiple-Interval-Scheduling}\ .$$

   Given input $(G = (V, E), k)$ for Independent-Set, create a valid input for Multiple-Interval-Scheduling. First, divide the processor time window into m distinct and disjoint *intervals* $i_1, \ldots, i_m$. Associate each interval i_j with an edge e_j. Next, create a different job J_v for each vertex $v \in V$. What set of time intervals should you pick for job J_v?

   (c) Finally, run the Multiple-Interval-Scheduling algorithm on the input you create, and output yes iff the Multiple-Interval-Scheduling algorithm outputs yes. Argue that the answer to Multiple-Interval-Scheduling gives you a correct answer to Independent-Set.

2. In the Four-Coloring problem, the input is a graph $G = (V, E)$, and you should output yes iff the vertices in G can be colored using at most four colors such that each edge $(u, v) \in E$ is *bichromatic*. Prove that Four-Coloring $\in$ NP-complete.

3. **Approximation Algorithm for Three-Coloring.** Recall the THREE-COLORING problem: Given a graph $G = (V, E)$, output YES iff the vertices in G can be colored using only three colors such that the endpoints of any edge have different colors. We saw in class that THREE-COLORING $\in$ NP-COMPLETE. Let THREE-COLORING-OPT be the following problem. Given a graph $G = (V, E)$ as input, color the vertices in G using at most three colors in a way that maximizes the number of *satisfied* edges, where an edge $e = (u, v)$ is satisfied if u and v have different colors.

   Give a deterministic, polynomial-time (3/2)-approximation algorithm for THREE-COLORING-OPT. Your algorithm must satisfy at least $2c^*/3$ edges, where for an arbitrary input $G = (V, E)$, c^* denotes the maximum number of satisfiable edges.

4. **(K&T 11.3)** Suppose you are given a set of positive integers $A = \{a_1, a_2, \ldots, a_n\}$ and a positive integer B. A subset $S \subseteq A$ is called *feasible* if the sum of the numbers in S does not exceed B:

$$\sum_{a_i \in S} a_i \leqslant B.$$

   The sum of the numbers in S will be called the *total sum* of S.

   You would like to select a feasible subset S of A whose total sum is as large as possible.

   For example, if $A = \{8, 2, 4\}$ and $B = 11$ then the optimal solution is the subset $S = \{8, 2\}$.

   (a) Here is an algorithm for this problem.

```
notQuiteRight(A = {a_1, ..., a_n}, B)
1  initialize S = ∅
2  define T = 0
3  for i = 1 to n:
4        if T + a_i ⩽ B
5              S ← S ∪ {a_i}
6              T ← T + a_i
7  return S
```

   Give an input for which the total sum of the set S returned by this algorithm is less than half the total sum of some other feasible subset of A. (You don't necessarily have to find the optimal subset, just *some* feasible subset.)

   (b) Give a polynomial-time approximation algorithm for this problem with the following guarantee: It returns a feasible set $S \subseteq A$ whose total sum is at least half as large as the maximum total sum of any feasible set $S' \subseteq A$. Your algorithm should run asymptotically faster than $O(n^2)$.