CSE 60111: Complexity and Algorithms
Homework 3

Reminder: For any algorithms question (unless otherwise specified), you must provide the following:

- an algorithm, meaning clear pseudocode, as well as perhaps a brief explanation of the pseudocode if you believe it helpful to clarify the code;
- an analysis of the runtime and space complexity of your algorithm;
- and a proof of correctness for your algorithm.

Each component is required and graded, so be sure you include all of them!

---

1. You have been asked to act as a consultant for the Port Authority of an ocean-side city. They're currently doing good business, and their revenue is constrained almost entirely by the rate at which they can unload the ships arriving in their port.

   Here's a basic sort of problem they face. A ship arrives, with n containers of weight $w_1, w_2, \ldots, w_n$. Standing on the dock is a set of trucks, each of which can hold up to K units of weight. (You can assume the w_i's and K are integers.) You can stack multiple containers in each truck, as long as you don't exceed total weight K on any one of them; the goal is the minimize the total number of trucks needed. (Note: This problem is NP-Complete, but you don't need to prove that fact.)

   A greedy algorithm for this might proceed as follows: Start with an empty truck, and begin piling containers $1, 2, 3, \ldots$ into it until the next container would overflow the capacity K. Now declare this truck loaded and send it off, and start loading the next truck. This algorithm, by considering trucks only one at a time, might not get the best total packing.

   (a) Give an example of a set of weights and a value of K where this algorithm does not use the minimum number of trucks.

   (b) Show, however, that the number of trucks used by this algorithm is within a factor of 2 of the minimum possible number, for any set of weights and any value of K.

2. 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: $\sum_{a_i \in S} a_i \leq B$. The sum of the numbers in S will be called the *total sum* of S. Your job is to select a feasible subset S of A whose total sum is as large as possible.

(a) Consider the following algorithm:

```
Let S ← ∅
Let T ← 0
For each i ← 1 . . . n
   If T + a_i ≤ B
        S ← S ∪ {a_i}
        T ← T + a_i
return S
```

Give an example where 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. (In other words, this is NOT a 2-approx.)

(b) Now, give a polynomial time approximation algorithm of this problem with the following guarantee: It returns a feasible set whose total some is at least half as large as the optimal feasible set. (In other words, find a 2-approximation.)

Hint: Your algorithm should be at most $O(n \log n)$ time, but you can probably do better.

3. Suppose we are given a collection of n jobs to execute on a machine containing a row of p identical processors. The parallel scheduling problem asks us to schedule these jobs on these processors, given two arrays $T[1..n]$ and $P[1..n]$ as input, subject to the following constraints:

- When the i^{th} job is executed, it occupies a contiguous interval of $P[i]$ processors for exactly $T[i]$ seconds.
- No processor works on more than one job at a time.

A valid schedule specifies a non-negative starting time and an interval of processors for each job that meets these constraints. Our goal is to compute a valid schedule with the smallest possible makespan, which is the earliest time when all jobs are complete.

Not shockingly, this problem is NP-Hard.

Describe a polynomial-time algorithm that computes a 3-approximation of the minimum makespan of a given set of jobs. That is, if the minimum makespan is M, your algorithm should compute a schedule with make-span at most 3M. You may assume that p is a power of 2. [Hint: you should definitely assume that p is a power of 2.]