CS386 Assignment 4

Due: Sunday 10/18 10PM ET

1. **Warm-ups** Do 2 of the following.

   (a) In Bin Packing we are given item sizes $s_1, \ldots, s_n > 0$ and a capacity B, and want to pack the items into as few bins of capacity B as possible. Show that, unless $\mathbf{P} = \mathbf{NP}$, no polynomial-time algorithm can guarantee that it uses fewer than $\frac{3}{2} \cdot \textsc{Opt}$ bins (that is, no algorithm has approximation ratio $< \frac{3}{2}$). (Show that you could use such an algorithm to correctly solve Partition.)

   (b) Recall the Knapsack FPTAS from class: given $\varepsilon > 0$, set $K = \varepsilon \cdot v_{\max}/n$, replace each value v_i by $v'_i = \lfloor v_i/K \rfloor$, and find the optimal solution for the *scaled* instance. Consider the instance below with $W = 10$ and $\varepsilon = 1/2$.

item	1	2	3	4
value	32	20	19	14
weight	5	3	3	3

      i. Find the optimal solution of the original instance and its value.

      ii. Compute K and the scaled values v'_i. Find the optimal solution of the scaled instance.

      iii. Is the scaled optimum also optimal for the original instance? What value does it actually achieve, and what does the FPTAS guarantee it must achieve at least? (The point of this was to address Tuan Anh's questions from class)

   (c) Let S^* be an optimal solution of a Knapsack instance (every single item fits on its own, $w_i \le W$), and let S' be an optimal solution for the scaled values $v'_i = \lfloor v_i/K \rfloor$ with $K = \varepsilon v_{\max}/n$. Fill in the justification for each step of the chain

   $$\sum_{i \in S'} v_i \overset{(1)}{\ge} K \sum_{i \in S'} v'_i \overset{(2)}{\ge} K \sum_{i \in S^*} v'_i \overset{(3)}{\ge} \sum_{i \in S^*} (v_i - K) \overset{(4)}{\ge} \textsc{Opt} - \varepsilon v_{\max} \overset{(5)}{\ge} (1-\varepsilon)\textsc{Opt}.$$

   (d) In Assignment 3 you (maybe, if you did the problem) gave an $O(nW)$ dynamic program indexed by weight. Here we need one indexed by *value*: let `B[`i`, `u`]` be the minimum weight needed to get at least u value using only the first i items.

      i. Write the recurrence for `B[`i`, `u`]` (take it to be ∞ if impossible), and say how to read the optimal scaled value off the table. (Mimic the include/exclude argument from Assignment 3.)

      ii. Show that after scaling, each $v'_i \le n/\varepsilon$, so u only needs to range up to n^2/ε. Conclude that the algorithm runs in time $O(n^3/\varepsilon)$, which is polynomial in both n and $1/\varepsilon$.

      iii. Why is this running time not a contradiction to Knapsack being **NP**-Complete?

   (e) Order the items so that $v_1/w_1 \ge v_2/w_2 \ge \cdots \ge v_n/w_n$ and assume each $w_i \le W$.

i. Show that the algorithm "take items in this order until the next one doesn't fit" can be arbitrarily bad: give, for every r, an instance where it returns a set of value less than $\textsc{Opt}/r$. (Two items suffice)

ii. Let k be the largest index such that $w_1 + \cdots + w_k \le W$ (assume $k < n$). Argue that $\textsc{Opt} \le v_1 + \cdots + v_k + v_{k+1}$. (Imagine you were allowed to take a *fraction* of item $k+1$ so the bag is exactly full; why can no integral solution beat the ratio-order fractional solution?)

iii. Conclude that returning the better of $\{1, \ldots, k\}$ and {the single most valuable item} has value at least $\frac{\textsc{Opt}}{2}$.

Do 2 and then either (4) or (3 and 5). That is, do: $2 \wedge (4 \vee (3 \wedge 5))$

2. **Hall's Marriage Theorem.** (We used the following theorem in our construction / proof of correctness of the 2/3 approximation protocol for MMS) Let $G = (L \cup R, E)$ be a bipartite graph. For $S \subseteq L$, let $N(S) \subseteq R$ denote the set of vertices adjacent to at least one vertex in S.

   Hall's Marriage Theorem says that there is a matching that covers every vertex in L if and only if

   $$|N(S)| \ge |S| \qquad \text{for every } S \subseteq L.$$

   (a) Prove the forward direction: if a matching covers every vertex in L, then $|N(S)| \ge |S|$ for every $S \subseteq L$.

   (b) Prove the reverse direction by induction on $|L|$. Assume $|N(S)| \ge |S|$ for all $S \subseteq L$. The claim is that there is a matching covering L.

   i. *Base case.* If $|L| = 1$, say $L = \{\ell\}$, why does a matching covering L exist?

   ii. *Inductive step.* Assume the theorem for all graphs with fewer than $|L|$ vertices on the left side, and split into two cases:

   - **Case 1:** for every *nonempty proper* subset $S \subsetneq L$, $|N(S)| \ge |S| + 1$ ("every proper subset has slack"). Pick any $\ell \in L$ and any neighbor r of ℓ (why does one exist?). Let G' be G with ℓ and r deleted. Show that G' satisfies Hall's condition, apply the inductive hypothesis, and add the edge $\{\ell, r\}$ back. (For $S \subseteq L \setminus \{\ell\}$, how many neighbors of S can have been lost when we deleted r?)
   - **Case 2:** some nonempty proper subset $S_0 \subsetneq L$ has $|N(S_0)| = |S_0|$ ("a tight set"). Consider two graphs:
     - G_1, induced on $S_0 \cup N(S_0)$. Show it satisfies Hall's condition. (For $S \subseteq S_0$, where do all of S's neighbors live?)
     - G_2, induced on $(L \setminus S_0) \cup (R \setminus N(S_0))$. Show it satisfies Hall's condition. (Take $T \subseteq L \setminus S_0$ and let $N_2(T)$ be its neighborhood in G_2, i.e. $N(T) \setminus N(S_0)$. Show $N(T \cup S_0)$ is the *disjoint* union of $N(S_0)$ and $N_2(T)$, then use Hall's condition for $T \cup S_0$ in G and the tightness of S_0.)

     Both graphs have fewer than $|L|$ vertices on the left (why?). Apply the inductive hypothesis to each and explain why the two matchings can be combined into a matching of G covering L.

   iii. Explain why Cases 1 and 2 cover all possibilities.

3. **Maximum Nash Welfare and MMS.** Consider an allocation instance with n players and indivisible goods M. Each player i has an additive, nonnegative valuation function v_i. The Nash welfare of an allocation $A = (A_1, \ldots, A_n)$ is

$$\mathrm{NW}(A) = \left(\prod_{i=1}^{n} v_i(A_i) \right)^{1/n}.$$

An allocation maximizing Nash welfare is called an *MNW allocation*. Assume throughout that the MNW allocation gives every player positive value. Write $\mathrm{MMS}_i = \mathrm{MMS}_i^{(n)}(M)$.

Last week's assignment mentioned that every MNW allocation gives each player at least a π_n fraction of their MMS, where

$$\pi_n = \frac{2}{1 + \sqrt{4n-3}}.$$

(a) What does this guarantee say when there are three players? What happens to the guarantee as n grows?

(b) Prove that every player i in an MNW allocation A receives value at least $\pi_n \mathrm{MMS}_i$. The proof below is split into steps; you may use that A is EF1 and Pareto optimal (as you may have shown in assignment 3).

Fix player i and write $a = v_i(A_i) > 0$. For each $j \neq i$, let g_j be the good in A_j that player i values most, and let $x_j = v_i(g_j)/a$. Finally let $\beta = \mathrm{MMS}_i/a$. We must show $\beta \leq 1/\pi_n = \frac{1+\sqrt{4n-3}}{2}$. If $\beta \leq 1$ we are done, so assume $\beta > 1$.

i. Show that for every $j \neq i$ with $v_i(g_j) > 0$,

$$v_i(A_j \setminus \{g_j\}) \leq \frac{a^2}{v_i(g_j)}.$$

Together with EF1 this gives $v_i(A_j \setminus \{g_j\}) \leq a \cdot \min\{1, 1/x_j\}$. (Write $b = v_j(A_j)$, $c = v_j(g_j)$, $y = v_i(g_j)$, $d = v_i(A_j \setminus \{g_j\})$. (1) Since A maximizes Nash welfare, moving g_j from j to i cannot increase the product: $(a+y)(b-c) \leq ab$. Deduce $c \geq \frac{by}{a+y}$. (2) Moving all of $A_j \setminus \{g_j\}$ from j to i cannot increase the product either: $(a+d)\,c \leq ab$. (3) Combine (1) and (2) and simplify to $dy \leq a^2$. Why did the other players' values not matter?)

ii. Let $T = \{j \neq i : x_j > \beta\}$, i.e. player i values g_j more than MMS_i, and let $t = |T|$. Explain why $t \leq n-1$ and why the goods g_j ($j \in T$) are distinct. Consider an MMS partition of M for player i into n bundles, each worth at least MMS_i. Argue that at least $n - t$ of the bundles contain no good g_j with $j \in T$, and so

$$v_i\big(M \setminus \{g_j : j \in T\}\big) \geq (n-t)\mathrm{MMS}_i = (n-t)\beta a.$$

iii. Split $M \setminus \{g_j : j \in T\}$ into A_i, the sets $A_j \setminus \{g_j\}$ for $j \in T$, and the sets A_j for $j \notin T \cup \{i\}$. Using (i):

- for $j \in T$, show $v_i(A_j \setminus \{g_j\}) < a/\beta$;
- for $j \notin T \cup \{i\}$, show $v_i(A_j) \leq a(\beta + 1/\beta)$. (Cases: $x_j \leq 1$ and $1 < x_j \leq \beta$. Note $x + 1/x \geq 2$ for all $x > 0$ and is increasing for $x \geq 1$.)

iv. Combine (ii) and (iii) to get

$$(n-t)\beta \leq 1 + \frac{t}{\beta} + (n-1-t)\Big(\beta + \frac{1}{\beta}\Big).$$

Simplify to $\beta \leq 1 + \frac{n-1}{\beta}$, i.e. $\beta^2 - \beta - (n-1) \leq 0$, and conclude $\beta \leq \frac{1+\sqrt{4n-3}}{2}$. Hence $v_i(A_i) \geq \pi_n \text{MMS}_i$.

4. **A PTAS for maximin shares.** In class we constructed a protocol that was $\frac{2}{3}$-MMS approximate (that is, every player has a strategy by which they can guarantee receiving at least $\frac{2}{3}$ of their own MMS of the total set of items). Unfortunately, the protocol asked each player to form the partitionings associated with their MMS and the best known exact algorithm for this is still exponential in time. Since we want reasonably quick protocols, we need a fast algorithm for players to use that can construct reasonable approximations of the partitioning associated with the MMS. Here you will construct the PTAS of Woeginger (1997) and prove it works.

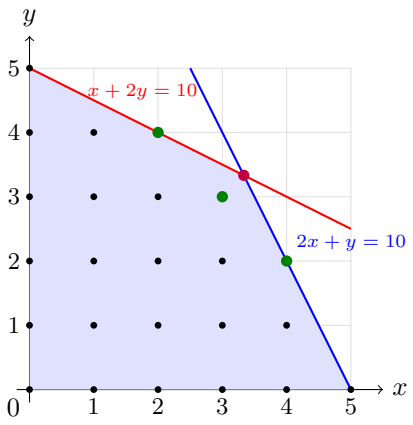
Necessary Background: Integer Linear Programs (ILPs). An *integer linear program* (ILP) is a problem where we are given the following as inputs:

- a set of (nonnegative) variables $x_1, ..., x_d$,
- a set of constraints, each of which is a linear inequality over the variables,
- and (optionally) a linear objective function: something to either minimize or maximize.

For example,

$$\begin{aligned} \text{maximize } & x + y \\ \text{subject to } & 2x + y \leq 10 \\ & x + 2y \leq 10 \\ & x, y \geq 0 \\ & x, y \in \mathbb{Z} \end{aligned}$$

Visually, this looks like:



Note that the green points are where the objective function is maximized.

The objective is to correctly output one of the following options:

- the set of constraints allows no feasible solutions,
- the objective function is unbounded with respect to the given constraints,
- or a feasible solution that maximizes the given objective function (e.g. $(2, 4)$, $(3, 3)$, or $(4, 2)$ as in the example above).

It turns out, ILPs can be quite difficult to solve, especially as more variables and constraints are introduced. Here's a decision version of the problem:

$$\text{ILP} = \left\{ \langle A, \mathbf{b}, \mathbf{c}, k \rangle \;\middle|\; \begin{array}{l} \text{there exists } \mathbf{x} \in \mathbb{Z}^d_{\geq 0} \text{ such that} \\ A\mathbf{x} \leq \mathbf{b} \text{ and } \mathbf{c}^\mathsf{T}\mathbf{x} \geq k \end{array} \right\}.$$

(a) First, determine A, $\boldsymbol{b}$, and $\boldsymbol{c}$ that would give us the example I gave above.

(b) Next, show that this problem is **NP**-Complete (literally every **NP**-Complete problem we've talked about this semester easily reduces to ILP: pick your favorite and do the reduction).

Very unfortunate that most interesting things in the world of computation are simply hard. However, if the input is 'small', there are some things we can do:

> **Theorem (Lenstra, 1983).** If the number of unknowns d is a fixed constant, then one can decide whether an integer program has a solution, and find one if so, in time polynomial in the number of bits needed to write down the constraints.

The theorem allows any number of constraints; in our use there will also be a constant number. **Note that I'm not asking you to derive this result - we are simply going to use it as a black box from here on out**. However, due to the flexibility and generality of ILPs, we will be spending quite a bit of time on them next semester in CS 486.

Okay so our plan is get this approximation algorithm to allow a given player to construct a partitioning that guarantees approximately their MMS. Throughout, $p_1, \ldots, p_n > 0$ are that player's (additive) values for the items, m is the number of bundles, $n \geq m$ (since otherwise the MMS is trivially 0), and

$$C^* = \max_{(S_1,\ldots,S_m)} \min_j \sum_{i \in S_j} p_i$$

is the player's MMS^m. Fix $\varepsilon \in (0, 1)$ and set $E = \lceil 1/\varepsilon^2 \rceil$ (an integer that is a *constant*, not part of the input). The goal: in time polynomial in n (for fixed ε), output a partition with every bundle worth at least $(1-\varepsilon)C^*$. The plan (starting from (c) because of subparts above): (c) a crude way to estimate C^* within a factor m; (d) the meat of the algorithm, an ILP for a special case; (e) a procedure $\textsc{Test}(C)$ that, given a guess C for C^*, either fails or outputs a partition with every bundle worth $\geq (1-\varepsilon^2)C$, and never fails when $C \leq C^*$; (f) a (fast) search over guesses.

(c) Imagine just processing the items in any order, putting each item into the bundle that currently has the smallest total (ties broken arbitrarily). Let $L_1 \leq L_2 \leq \cdots \leq L_m$ be the final bundle valuations. Recall that we are trying to maximize the value of the smallest bundle L_1: let's show $C_0 := L_1$ satisfies $C_0 \leq C^* \leq mC_0$.

i. Why is $C_0 \leq C^*$? (This should be obvious by definition of C^*)

ii. Explain why every bundle has at least 1 item.

iii. Index the bundles so that bundle j ends with total L_j, and for $j \geq 2$ let r_j be the last item placed in bundle j. Explain why the total value of bundle j just before r_j was added is *at most* C_0. (When r_j was placed, its bundle was the smallest one. Compare with the bundle that ends with total C_0, and remember totals only grow.)

iv. Let $R = \{r_2, ..., r_m\}$ and consider the set $\bar{R}$ of all items not in R. Why must the total value of $\bar{R}$ be at most mC_0 (look at iii)

v. Let $B_1^*, ..., B_m^*$ be some optimal solution (so each B_i^* has value at least C^*). Explain why there must be some B_k^* without a single item of R. (Obvious)

vi. Put it together. Why must $C^* \leq mC_0$? (iv + v)

Note that immediately we now have something like a $\frac{1}{m}$ approximation algorithm for MMS. Obviously this isn't great and certainly alone will not get us our goal of being arbitrarily close to the correct answer. The rest of the algorithm basically 'rescales' and 'trims down' the input to make the above approximation work better.

(d) Suppose we are given $m \geq 1$ and numbers $a_1, \ldots, a_r$ with $0 < a_i \leq 9E^2 = 9\left(\lceil 1/\varepsilon^2 \rceil\right)^2$, and promised that

($\star$) the numbers can be split into m parts each of total at least $9E^2$.

In a zoomed-out picture, we're going to construct an ILP satisfying:

- The number of variables will be *constant* (so that we can apply Lenstra's Theorem above)
- The result of the variables will tell us how to build a partitioning that guarantees each bundle has value at least $9(E^2 - E)$.

The rough idea is to separate out the 'big' and 'small' items in $\{a_i\}$ to make life a little easier to analyze. Specifically, we'll call jobs *small* if their value is smaller than $3E$, and *big* otherwise. As they carry the most weight and largely define the structure of the partitions, we'll mostly focus on where the big numbers end up. We first introduce some notation for describing the big jobs assigned to a machine. For a multiset X of such goods, define its *type* $t(X)$ to be the vector

$$t(X) = (t_{3E}, t_{3E+1}, \ldots, t_{9E^2}),$$

where t_j denotes the number of elements of X whose values are in $[j, j+1)$ (so $t_j = |\{x \in X \mid j \leq x < j+1\}|$). The type records the sizes and multiplicities of the goods, but not their identities.

For example, suppose $E = 1$ and let

$$X = \{3.4, 3.6, 4.5, 5.0\}, \qquad Y = \{5.3, 7.1, 7.4, 8.5, 9.0\}.$$

Using the type vector $(t_3, t_4, \ldots, t_9)$ (recall $3E = 3$ and $9E^2 = 9$), we obtain

$$t(X) = (2, 1, 1, 0, 0, 0, 0), \qquad t(Y) = (0, 0, 1, 0, 2, 1, 1).$$

In particular, two multisets have the same type precisely when they contain the same number of rounded values of every size.

We define the *size* of a type t by

$$\textsc{Size}(t) = \sum_{j=3E}^{9E^2} j t_j.$$

This is an *under-estimate* of the total value of goods represented by the type. For the illustrative example above,

$$\textsc{Size}(t(X)) = 2 \cdot 3 + 1 \cdot 4 + 1 \cdot 5 + 0 \cdot \sum_{j=6}^{9} = 6 + 4 + 5 = 15$$

Now consider enumerating all types whose total size is at most $18E^2$. Let $\mathcal{T}$ be the set of these types, and let $T = |\mathcal{T}|$ denote the number of types. We write the enumeration as $\mathcal{T} = \{t^{(1)}, t^{(2)}, \ldots, t^{(T)}\}$. The superscript (i) is an index, not an exponent: $t^{(i)}$ denotes the ith type in the enumeration. The ordering is arbitrary.

More explicitly, every type in $\mathcal{T}$ is a vector $t^{(i)} = \left(t^{(i)}_{3E}, t^{(i)}_{3E+1}, \ldots, t^{(i)}_{9E^2}\right)$, where each coordinate $t^{(i)}_j$ is a nonnegative integer and

$$\textsc{Size}(t^{(i)}) = \sum_{j=3E}^{9E^2} j\, t^{(i)}_j \leq 18E^2.$$

The empty type, whose coordinates are all zero, is included in this enumeration.

As in the example above, letting $E = 1$, here are some of the types whose size is at most $18E^2 = 18$.

Type t	Rounded (Multi)-Set	$\textsc{Size}(t)$
$(0,0,0,0,0,0,0)$	$\emptyset$	0
$(6,0,0,0,0,0,0)$	$\{3,3,3,3,3,3\}$	18
$(5,0,0,0,0,0,0)$	$\{3,3,3,3,3\}$	15
$(4,0,0,1,0,0,0)$	$\{3,3,3,3,6\}$	18
$(4,0,1,0,0,0,0)$	$\{3,3,3,3,5\}$	17
$(4,1,0,0,0,0,0)$	$\{3,3,3,3,4\}$	16
$(3,0,0,0,0,0,1)$	$\{3,3,3,9\}$	18
$\vdots$	$\vdots$	$\vdots$

Because E is fixed throughout this section, both the number of possible coordinates and the upper bound on the total size are **constants**. Consequently, the number T of types depends only on E, not on the number of goods n or bundles m. It might be a really big number, but who cares! It is only a function of E.

For our last definition, given a type $t = (t_{3E}, ..., t_{9E^2})$, it will be helpful to know roughly how many 'small' numbers are needed to 'complete' a bundle of that type, ie how many smalls we need to add to get to $9(E^2 - E)$. To ensure we get an overestimate, we'll be a little generous and set $c(t) = \max\{0, 9E^2 - 6E - \textsc{Size}(t)\}$. So $c(t)$ is an overestimate of the total value of smalls needed to get a bundle whose type is t to total value $9(E^2 - E)$.

Now let's build the ILP. For each type $t^{(i)}$, introduce a nonnegative integer variable x_i. The intended meaning of x_i is the number of bundles whose assigned rounded big values have type $t^{(i)}$.

For example, if $x_i = 3$ and $t^{(i)} = (2,0,1,0,0,0,0)$ in the $E = 1$ example, then three bundles each receive two goods of rounded value 3 and one good of rounded value 5. Hence the variable x_i counts bundles, not goods.

The integer linear program will choose the values $x_1, \ldots, x_T$. Its first constraint,

$$\sum_{i=1}^{T} x_i = m,$$

ensures that all m machines are accounted for. The remaining constraints ensure that the selected types do not require more big jobs of any size than are available, and that the available small jobs suffice to complete the machines to the required minimum total size.

Consider the following integer program:

$$\begin{aligned}
&(1) \quad \textstyle\sum_i x_i = m, \\
&(2) \quad \textstyle\sum_i c(t^{(i)})\, x_i \le S, \\
&(3) \quad \textstyle\sum_i t^{(i)}_j\, x_i \le n_j \quad \text{for } j = 3E, \ldots, 9E^2,
\end{aligned}$$

where S is the sum of all small numbers and n_j is the number of big a_i with $\lfloor a_i \rfloor = j$. Note that there's no objective function: we actually don't really care about maximizing anything here, we just want *any* feasible solution.

i. Explain what each of (2) and (3) enforce.

ii. Take a partition promised by ($\star$) (2 pages ago, lol) into parts $P_1, \ldots, P_m$. While some part has total $> 18E^2$, delete one element from it. Show that every part ends with total in $[9E^2, 18E^2]$.

iii. Call the trimmed parts P'_k. Let t_k be the type of the rounded big numbers of P'_k. Show:
   - P'_k has at most $6E$ big numbers, so $\text{BIG}(P'_k) \le \text{SIZE}(t_k) + 6E$, where BIG is the total of the *unrounded* big numbers;
   - $\text{SMALL}(P'_k) \ge 9E^2 - \text{BIG}(P'_k) \ge c(t_k)$.

iv. Conclude that $x_i = |\{k \mid t_k = t^{(i)}\}|$ satisfies (1), (2), (3) of the above ILP. (For (2), sum the second bullet over all k; the small numbers in different P'_k are disjoint.)

v. Show how to build m parts, using (1) and (3) to assign actual big numbers to x_i parts of type $t^{(i)}$ (each big number is used at most once; leftover big numbers go in a leftover pile Q_0). Then, for each part $k = 1, \ldots, m$ in turn, add small numbers one at a time until its total is at least $9(E^2 - E)$. Show that
   - the small numbers added to part k total at most $c(t_k)$, where t_k is its type; (Before the last small number is added the total is $< 9(E^2 - E)$; the part's big numbers total at least $\text{SIZE}(t_k)$; the last small number is $< 3E$.)
   - by (2) we never run out of small numbers;
   - putting all leftovers (Q_0 and unused small numbers) into part 1 leaves every part with total $\ge 9(E^2 - E)$.

vi. Conclude that this special case (where each value is at most $9E^2$ and the values can be split into m parts each of total at least $9E^2$) can be solved in polynomial time. (We already identified that the size of the ILP is technically constant - but what parts do we actually need to do that are dependent on n and m?)

Okay, so we can form an ILP if the instance is nice and pass it to the algorithm suggested by Lenstra's theorem. Now, we need to make these nice instances. To do that, we'll come up with a nice instance if we have a reasonable 'guess' of the correct value of C^*. That is, given some guess C, we'll return bundles such that every bundle has value at least C, or report that we couldn't find one.

(e) Given such a guess $C > 0$: First, let $a_i = 9E^2 p_i / C$. Let k be the number of i with $a_i > 9E^2$. If $k \ge m$, give m of those items one bundle each, dump everything else anywhere, and succeed. Otherwise, make each such item its own bundle. Run the special-case algorithm from (d) on the remaining numbers (all $\le 9E^2$) with $m - k$ parts. If it succeeds, output the m bundles; if the IP is infeasible, fail. (To "fail", the algorithm may declare the IP infeasible; ($\star$) is a promise that need not hold in general.)

i. Show that whenever $\textsc{Test}(C)$ succeeds, every bundle it outputs has value at least $(1-\frac{1}{E})C \geq (1-\varepsilon^2)C$. (You know that in the scaled instance each bundle has value at least $9(E^2-E) = 9E(E-1)$)

ii. Show that if $C \leq C^*$, then $\textsc{Test}(C)$ succeeds. (Take an optimal partition; scaled, each part is worth $\geq 9E^2$. Let k be the number of large items (those with $a_i > 9E^2$). At least $m-k$ parts contain no large item; why? Remove the large items from the picture; merge the leftover parts into the $m-k$ parts to show $(\star)$ holds for the remaining numbers with $m-k$ parts. Use (d).)

iii. Explain why if $\textsc{Test}(C)$ succeeds then $C^* \geq (1-\varepsilon^2)C$. Equivalently, if $\textsc{Test}(C)$ fails then $C > C^*$.

(f) **Searching for the guess.** Let $\alpha_i = (1+\varepsilon)^i C_0$ for $i = 0, 1, 2, \ldots$

i. Show $\textsc{Test}(\alpha_0)$ succeeds, and that $\textsc{Test}(\alpha_i)$ fails for some $i \leq \log_{1+\varepsilon}\big(m/(1-\varepsilon^2)\big) + 1 = O(\frac{1}{\varepsilon}\log m)$. (Use (e).iii. and $C^* \leq mC_0$.)

ii. Let k be such that $\textsc{Test}(\alpha_k)$ succeeds and $\textsc{Test}(\alpha_{k+1})$ fails (try $i = 0, 1, \ldots$ until the first failure). Show that the partition output by $\textsc{Test}(\alpha_k)$ has every bundle worth at least $(1-\varepsilon)C^*$. (By (e).iii., $\alpha_{k+1} > C^*$; and $(1-\varepsilon^2) = (1-\varepsilon)(1+\varepsilon)$.)

(g) Holy moly, we did it. Some final thoughts:

- State the total running time for fixed ε (the constant depends on T). Where does the exponential dependence on $1/\varepsilon$ come from?
- The difference between a PTAS and an FPTAS is that a PTAS needs to have a polynomial runtime in the size of the input **when we treat ε as a constant**; an FPTAS needs to be not just polynomial in the input size **but also polynomial in $\frac{1}{\epsilon}$**. Explain why this is a PTAS but not an FPTAS.

5. **The 2/3-MMS protocol.** In class, we studied the protocol from Procaccia and Wang's paper that guarantees each player at least 2/3 of their MMS. Recall that MMS is Maximin Share and that player i's MMS of a given set of items $\mathcal{M}$ is determined looking over all partitionings $\Pi^N(\mathcal{M}) = \{(S_1, ..., S_N) \mid S_i \cap S_j = \emptyset \text{ for all } i \neq j \text{ and } \cup_i S_i = \mathcal{M}\}$ and is defined by

$$\text{MMS}_i^N(\mathcal{M}) = \max_{\Pi^N(\mathcal{M})} \min_j v_i(S_j)$$

(a) Recall the following term from class:

$$\rho_N = \max\{\lambda \mid \forall \mathcal{M},\ \forall S \subseteq \mathcal{M},\ \forall v_i \text{ (additive)},\ \forall n \leq N : \\ v_i(\mathcal{M} \setminus S) \leq (N-n)\lambda \text{MMS}_i^N(\mathcal{M}) \implies \text{MMS}_i^n(S) \geq \lambda \text{MMS}_i^N(\mathcal{M})\}$$

The paper shows that $\rho_N \geq 2/3$ with a long argument. We'll be gentler: show that $\rho_N > \frac{1}{2}$, by showing that $\lambda_N = \frac{N}{2N-1}$ satisfies the implication (so $\rho_N \geq \lambda_N > \frac{1}{2}$). Fix $N \geq 2$, $n \leq N$, S, and $R = \mathcal{M} \setminus S$, and drop the subscript i (we'll just focus on player i, so it won't be ambiguous). Let $\mu = \text{MMS}^N(\mathcal{M})$; assume $\mu > 0$ and $v(R) \leq (N-n)\lambda_N \mu$.

i. Take an MMS partition $P_1, \ldots, P_N$ of $\mathcal{M}$, so $v(P_k) \geq \mu$ for every k. Call P_k *damaged* if $v(P_k \cap R) > (1-\lambda_N)\mu$ and *intact* otherwise. Show that if P_k is intact then $v(P_k \cap S) \geq \lambda_N \mu$.

ii. Let c be the number of damaged bundles. Show that $c = 0$, or else $c(1-\lambda_N)\mu < v(R)$. Conclude $c < (N-n) \cdot \frac{\lambda_N}{1-\lambda_N} = (N-n) \cdot \frac{N}{N-1}$. (The $P_k \cap R$ are disjoint subsets of R.)

iii. Show $(N-n)\cdot\frac{N}{N-1}\le N-n+1$. (Write $\frac{N}{N-1}=1+\frac{1}{N-1}$ and use $n\ge 1$.) Since c is an integer, conclude $c\le N-n$, so at least n bundles are intact.

iv. Build an n-partition of S with every bundle worth at least $\lambda_N\mu$ from the intact bundles. Conclude $\mathrm{MMS}^n(S)\ge\lambda_N\mu$.

(b) The protocol in class is not technically a polynomial time protocol. In particular, we ask a player in each round of the algorithm to split the items into a MMS partitioning. Since this is known to be **NP**-Hard to do, it's a bit unfair to ask them that. However, there is a nice PTAS (Woeginger 1997, problem 4 above): given $\varepsilon>0$, in time polynomial in the number of items (for fixed ε) we can find a partitioning $\pi\in\Pi^n(S)$ such that $\min_j v_i(\pi_j)\ge(1-\varepsilon)\mathrm{MMS}_i^n(S)$.

Let λ be any number satisfying the implication in the definition of ρ_N (for instance λ_N from (a), or ρ_N itself). Recall that Apx-MMS$(X,N,S,\mathcal{M})$ (here X is the set of players still waiting for a bundle, S the items still unallocated, and we write $n=|X|$) is called with the invariant

$$(*)\qquad v_i(\mathcal{M}\setminus S)\le(N-|X|)\lambda\,\mathrm{MMS}_i^N(\mathcal{M})\quad\text{for all } i\in X,$$

and does: (1) pick some player $p\in X$, who computes an n-maximin partition $S_1,\dots,S_n$ of S; (2) build the bipartite graph with left vertices X, right vertices $Y=\{S_1,\dots,S_n\}$, and an edge (k,j) whenever $v_k(S_j)\ge\lambda\,\mathrm{MMS}_k^N(\mathcal{M})$; (3) compute X^+ (the largest $Z\subseteq X\setminus\{p\}$ with $|Z|\ge|f_Y(Z)|$), $Y^+=f_Y(X^+)$, $Y^-=Y\setminus Y^+$ and $X^*=X\setminus X^+$ as in class; (4) match X^* into Y^- via Hall and give those bundles away; (5) recurse on X^+ with the leftover items.

Modified protocol. Before anything else, every player i runs the PTAS from problem 4 on $(\mathcal{M},N)$ and records m_i, the value of the smallest bundle in the partition it found, so $(1-\varepsilon)\mathrm{MMS}_i^N(\mathcal{M})\le m_i\le\mathrm{MMS}_i^N(\mathcal{M})$. Set $\tau_i=(1-\varepsilon)\lambda\,m_i$. In step (1), player p instead computes a partition with $\min_j v_p(S_j)\ge(1-\varepsilon)\mathrm{MMS}_p^n(S)$, and in step (2) the edge (k,j) is present when $v_k(S_j)\ge\tau_k$. Everything else is unchanged; the invariant $(*)$ is the *same* as before.

i. Show that $(1-\varepsilon)^2\lambda\,\mathrm{MMS}_i^N(\mathcal{M})\le\tau_i\le(1-\varepsilon)\lambda\,\mathrm{MMS}_i^N(\mathcal{M})$. Why can player i actually compute τ_i in polynomial time, whereas it could not compute $\lambda\,\mathrm{MMS}_i^N(\mathcal{M})$?

ii. Assume $(*)$. Show that in step (2) it follows that player p satisfies $v_p(S_j)\ge\tau_p$ for every j, so (p,j) is an edge for all j. (Use $(*)$ and the definition of λ (with this $n=|X|\le N$) to bound $\mathrm{MMS}_p^n(S)$ from below, then the PTAS guarantee, then part (i). You can assume they 'played honestly'.)

iii. Look back at the proof from class that showed that for all $Z\subseteq X^*$ it held that $|Z|\le|f_{Y^-}(Z)|$ (the condition by which we applied Hall's theorem). Which property of the graph did the proof use about player p, and which property about the definition of X^+? Explain why neither depends on which threshold defines the edges, so the matching exists.

iv. Let $Y^*\subseteq Y^-$ be the matched bundles and S^* the unmatched items. Show that for every $k\in X^+$,

$$v_k(\mathcal{M}\setminus S^*)\le\big(N-|X^+|\big)\lambda\,\mathrm{MMS}_k^N(\mathcal{M}),$$

so $(*)$ holds in the recursive call. (For $k\in X^+$ and $j\in Y^*$, there is no edge (k,j), so $v_k(S_j)<\tau_k$. Compare τ_k with $\lambda\mathrm{MMS}_k^N(\mathcal{M})$ using (i), then follow the computation from the lecture notes.)

v. Show that for $|X| = 1$ the single remaining player p gets value at least τ_p. ($v_p(\mathcal{M}) \geq N\,\mathrm{MMS}_p^N(\mathcal{M})$.) Conclude by induction that every player i receives at least $\tau_i \geq (1-\varepsilon)^2\lambda\,\mathrm{MMS}_i^N(\mathcal{M})$. Also check that $(*)$ holds in the very first call, Apx-MMS$([N], N, \mathcal{M}, \mathcal{M})$.

vi. Take $\lambda = \lambda_N = \frac{N}{2N-1}$ and $\varepsilon = \frac{1}{4N}$. Show that every player receives at least $\frac{1}{2}\mathrm{MMS}_i^N(\mathcal{M})$. (Use $(1-\varepsilon)^2 \geq 1 - 2\varepsilon$.) (With the true ρ_N, which is at least $\frac{2N}{3N-1} = \frac{2}{3}\left(1 + \frac{1}{3N-1}\right)$, the paper gets 2/3 with $\varepsilon = \Theta(1/N)$ the same way.)

vii. For a constant number of players N, argue that the protocol runs in polynomial time in the number of items: how many PTAS calls are made in total, how long does it take to compute X^+ (brute force over subsets of $X \setminus \{p\}$), and how long to find the matching in step (4)? Where would the argument break if N were part of the input?