Team Tingxiu
Pinxiu Gong 25404900
Ting Ding: 26253690

Input files:

When we generate the input, we randomly generate a P and an M between 2^20 and 2^32. We use 200,000 items. We randomly generate betweem 1000 and 10,000 constraints. We thought random would be better to throw people off the most, since people can't possibly think of a strategy for every possible situation.

For each item, we generate a random class index between 0 and N, a random weight between 0 and P, a random cost between 0 and M, and a random resale value between 0 and 2^32. We kept adjusting the values of P, M and N until the output file is approximately 4MB.

For each constraint, we randomly choose a start index between 0 and N-2, and a random length between 2 and N-start index. Then we include all the class indexes between the start index and the end index, which is start+length. We make these classes incompatible with each other.

Algorithm:

We actually used a variety of methods and then took the best inputs from all of them. We have two main varieties: in one which we will call **class-method**, we group by class and take ALL the items from each class we include. In the other, which we will call **item-method**, we just take all the incompatible pairs.

We used a greedy algorithm where we first prune the items where cost is lest than resale value, because these are sold at a loss so we never want to include them in our bag. We also take out all items whose cost is greater than the money we have and items whose weight exceeds that of our bag, because these two types are silly to include since they either explode the bag or deplete all of our money right away.

We use three different heuristics and take the max outcome of using each one:

1.) The first heuristic is (cost*weight)/(1+(resale-cost)**2). We multiply cost and weight in the numerator because we want to minimize both. In the denominator we include the term 1 to prevent zero values. Then we subtract resale from cost to get the net income, and square it because we want to balance out the TWO terms in the numerator, and also because it yielded better results in our runs.
2.) In the second heuristic, we want weight to be balanced out by net income and cost to be balanced out by resale
3.) In the last heuristic, we account for weight and cost relatively equally based on the proportion to the heaviest item/costly item.

Then, we run 12 scripts, each using a different heuristic. In 9 of the strategies we also included some amount of randomization according to some margin. The randomization occurs in the way we choose items (in **item-method**) and the way we choose classes (in **class-method**).

In the scripts themselves, we first sort based on the heuristic. We then "classify" the items, by initializing each class as empty and then putting into each class the items that are in that class. We do this because we use a strategy where, in an attempt to circumvent the complexity of avoiding putting incompatible classes together, we sort all the classes and we just take all the items from each class in the order that we sort them (unless they are incompatible in which case we skip that class) (or our randomization threshold is met and we just choose classes randomly). Note: we only use the previously mentioned class-method if there are enough items in each class. If each class only has around two items in it, then we don't group by class, because then the overhead is much greater than the work avoided. After using this strategy, we jump to **GargSack Putting Method.**

In the case that we don't have enough items in each class, we just look at each of the items individually. To prevent the GargSack from exploding, we initialize a set of incompatible classes. It performs like a pseudo-adjacency matrix such that if items of class i and j are incompatible, then the pairs (i,j) and (j,i) will be in the "incompatibles" set. We include both in order to avoid the overhead of checking the list twice every time we want to verify an incompatibility. Taking our sorted list (unless our randomization threshold is met in which case we just insert randomly and ignore the sorting), we pull each item based on priority. Then, we insert them into a "potential GargSack candidates" list based on whether or not that item is compatible with the items already in the list. After we obtain this list of items that **are** all compatible with each other, we move on to **GargSack Putting Method.**

GargSack Putting Method → After getting the list of potential candidates, we proceed to actually put the elements in the bag (real_list) in order by the candidates list (unless our randomization threshold is met in which case we just pick items randomly) by incrementing the weight of the GargSack, decrementing our available money, and increasing our gross profit. We do this for each item in the candidates list. If we come across an element that exceeds the weight of the GargSack or puts us in debt, then we skip that item and don't put it in. We continue looking at the rest of the elements. After maxing out either the weight of the GargSack or our cash, we calculate our net income which is just our gross profit + initial amount of money – total cost. We then return all the elements in real_list.

Our actual output files are an amalgamation of our best outputs from all the times we ran our code and all the strategies we used.