Spring 2008

Heuristics for the Knapsack Problem

Claire Gilbertson

Carroll College, Helena, MT

Follow this and additional works at: https://scholars.carroll.edu/mathengcompsci_theses

Part of the Applied Mathematics Commons, and the Mathematics Commons

Recommended Citation


https://scholars.carroll.edu/mathengcompsci_theses/56
Heuristics for the Knapsack Problem

by
Claire Gilbertson

An honors thesis submitted to the
Department of Mathematics, Engineering,
and Computer Science in partial fulfillment
of the requirements for graduation with honors.

Carroll College
April 2008
This thesis for honors recognition has been approved for the Department of Mathematics, Engineering, and Computer Science

Dr. Holly Zullo

Date 4/15/08

Dr. Mark Parker

Date 4/16/08

Dr. Jeff Morris

Date 4/16/08
Abstract

We explore the knapsack problem where the goal is to maximize the value of packed objects for a certain container. The knapsack problem is NP-complete which means the time needed to solve it exactly grows exponentially as the size of the data set increases. Due to the infeasibility of using algorithms which find all possible solutions, we can use heuristics which are methods used to estimate a solution rapidly. We create a Java program to test seven heuristics on 21 data sets. The seven packing algorithms include four basic greedy algorithms, two backtracking methods, and one random packer. We use data sets from a standard library and create our own data sets with specific characteristics. The greedy algorithm which sorts by largest utility and the one of the backtracking methods perform the best.
Table of contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>2</td>
</tr>
<tr>
<td>2. Data Sets</td>
<td>4</td>
</tr>
<tr>
<td>3. Packing Algorithms</td>
<td>6</td>
</tr>
<tr>
<td>4. Implementation</td>
<td>8</td>
</tr>
<tr>
<td>5. Results</td>
<td>11</td>
</tr>
<tr>
<td>6. Conclusions and Future Works</td>
<td>27</td>
</tr>
<tr>
<td>7. References</td>
<td>28</td>
</tr>
<tr>
<td>8. Appendix</td>
<td>29</td>
</tr>
</tbody>
</table>
Chapter 1: Introduction

We explore the knapsack problem using a variety of basic heuristics and data sets. The packing methods or algorithms are implemented using Java programs compiled on BlueJ. The data sets used have both known and unknown optimal solutions.

Problem Overview

The knapsack problem is a packing problem in which the goal is to maximize the value of packed objects for a certain container. For example, if a student has numerous things to bring to school, the student has to choose which objects are most valuable and how many can fit in the “knapsack.” Perhaps the backpack only holds 50 pounds and has a maximum volume of 1 cubic foot. Does he bring his math book to school which weighs a lot and takes up a large portion of the available space of his backpack? Or does he only bring the books needed for classes that have assignments?

The knapsack problem aims to pack these objects in such a way as to find the set of items which yields the greatest value. The most general form of the knapsack problem is multidimensional, meaning that it allows for multiple constraints such as weight and volume. For this study, we limited the scope to one-dimensional problems due to the limitations of time and the complexity of multidimensional problems. For ease of nomenclature we will refer to constraint values as a “weight” and an item’s value as a “utility.” These could refer to other characteristics, such as volume or price but is inconsequential for this study.

The knapsack problem is NP-complete, which means the time needed to solve it exactly grows exponentially as the size of the data set increases. There is no known polynomial algorithm which is best for choosing items to be placed into a certain container. “NP-complete” stands for “non-deterministic polynomial time,” and a solution to any NP-complete problem would solve all NP problems. Potential solutions are easy to check for feasibility by verifying they satisfy all constraints. Due to the intractability of using algorithms which find all possible solutions, we can use heuristics which are methods used to find a “good” solution rapidly. There are numerous ways of “packing” objects: greedy algorithms which either pack the highest utility or lightest objects first, look-ahead algorithms which compare objects out of order, and many others.

Formal description of the 1-d knapsack problem

Objective Function to maximize: \( \sum_{i=1}^{n} v_i x_i \)

Subject to: \( \sum_{i=1}^{n} c_i x_i \leq W \)

Where

- \( n \): number of items in the problem set
- \( i \): index that refers to an item in the problem set
- \( W \): maximum total weight
- \( x_i \): \( i \)th item in set, binary variable such that \( \begin{cases} 0 & \text{if item } i \text{ not in knapsack} \\ 1 & \text{if item } i \text{ in knapsack} \end{cases} \)
- \( c_i \): is the weight of the \( i \)th item
- \( v_i \): is the utility of the \( i \)th item
Applications

The knapsack problem applies to numerous situations such as cargo loading, cutting stock, production scheduling, project selection, capital budgeting, and portfolio management.

An example of the knapsack problem is loading cargo into a container, such as railroad cars or semi trucks. Each container has a variety of constraints such as shape (height and length) and weight capacity. Also each piece of cargo has properties such as weight, volume, shape, monetary worth, etc. For example, frozen peas might be packed in 1 cubic foot containers that weigh 15 pounds each. The machines or manpower needed to pack the cargo also has characteristics to consider. For example, a fork lift needs fuel, maintenance, a driver, special equipment to load different items, etc. The goal for cargo loading could be to maximize profit. In this case the objective function is to maximize profit. This is a multidimensional problem and the weight constraints come from the container: height, length, and weight capacity. The items are the pea containers with the following constraints: volume, weight, and cost to load into the container.

Another example of the knapsack problem is capital budgeting. Heuristics can be used to pick new investments for a company such as whether or not to implement new machines, buy a certain product, and even undertake research and development projects. The objective function is to maximize the company’s profit or value to its stockholders by picking the “best” investments. The objects in this case are the investments, and the constraints for the investments are the limits of resources the company wants to allocate for the investments. The net present value can be used as the value of each object (investment), and the initial financial cost of the investment, return time of the investment, and other risk factors to the company are the constraints for the problem.

The following is a comical example of the knapsack problem ([3]):

“Farmer John's favorite hobby is creating a tape containing some of Bessie's favorite music to listen to while she's being milked. The amount of milk that Bessie produces is dependent on the songs that Bessie listens to while being milked. Given a collection of songs, each represented by a pair of integers, the length of the song (in seconds), the amount of milk Bessie produces while she's listening to that song, and the total amount of time that it takes to milk Bessie, find the set of songs such that their total length is no more than the time it takes to milk Bessie and they maximize Bessie's milk production.”

There are numerous applications of the knapsack problem and not all are as obvious as the cargo loading situation. The goal behind all knapsack problems is to pack the “best” items from a set to maximize the return of the items.
Chapter 2: Data Sets

For the knapsack problem there are numerous data sets with known optimal solutions on which we can perform tests and experiment with heuristics to pack items efficiently. We generated some original data sets specifically designed to use with certain packing algorithms that we created. This allowed us to compare the relative effectiveness of certain heuristics on different types of data sets. Additionally we used several mknap files from the OR-Library maintained by J. E. Beasley ([1]). We modified the mknap test sets to make them one-dimensional. The files used to store the problem information are basic data files modeled after the mknap problem format. The first entry is the number of problems the particular data file contains. The second row has three values: the number of objects, the number of constraints (always 1), and the optimal objective function value. The third row contains the utility of each object, and the fourth row represents the weight of each object. The last row represents the maximum allowable weight. The example below has 6 objects, 1 constraint, and an optimal solution of 3800.

We generated various types of data sets and used existing data sets specifically created for use with the knapsack problem. We used Excel to create the data sets which are characterized in Chart 1.

Small data sets
- Fewer than 25 items per data set
- Created data sets by hand
- Hand-picked data sets such that optimal solution is found using a certain algorithm
- Useful because optimal solution is easily found
- Used to test computer code for accuracy of programs created
- Data sets included in this category: 1, 2, 3, 4, 5, 8, 12, 17, 18

Large data sets
- More than 25 items per data set
- Generated random numbers in Excel for the object utilities and weights
- Compare algorithms’ effectiveness with small data sets
Much more difficult to solve exactly
Data sets included in this category: 6, 7, 9, 10, 11, 13, 14, 15, 16, 19, 20, 21

Clumps

- Generated 50 items of which sets of ten items had similar (or “clumped”) utilities and/or weight values
- Used Excel’s random number generator
- Data sets included in this category: 11, 13, 14, 16

Additionally we picked data sets so that certain packing methods would find the optimal solution or come closest to the actual value. We used these data sets to show the effectiveness of certain algorithms on various data sets.

Sets where specific methods work best or are designed to fail

Some sets are designed so that certain packing methods will always work better than other methods. For example, data set 15 is always packed best by a method which sorts by a greedy algorithm based on largest utility or smallest weight. This is due to the nature of the data set which has corresponding weights and utilities. (see appendix A, data set 15). Heuristics which first pack small utilities or large weights first always fail on this data set. On the other hand, data set 14 has very large weights for greater utility values (see appendix A, data set 14) and the solution is approximated very poorly by packing the largest utility first. These sets are useful for comparing heuristics because they model real life situations, and a heuristic which performs better on average on a variety of data sets can be more useful than a heuristic which approximates one type of data set very well, but fails at other sets.

Chart 1: Description of the data files(131,689),(869,868)
Chapter 3: Packing Algorithms

Multiple packing algorithms were created to choose items from a set. We began by creating basic greedy algorithms, and built on them to include various backtracking methods. For this exploration we limited the scope of the study to using data sets with items of one utility and weight, so if data sets had multiple weights per object, such as the mknaps files, we used the first weight listed for analysis. See appendix A for each data set.

1. **Random packer: chooses items randomly**
   This algorithm chooses items using a random number generator. The items are packed while the sum of their weights is less than or equal to the maximum weight. This heuristic gives a basis for this study; we aim to create algorithms which pack objects of greater overall utility than that of a random packer.

2. **Greedy Algorithms**
   Greedy heuristics are a simple method of choosing items; the items are ordered by weight or utility and packed consecutively while keeping under the maximum allowable weight. We created the following four greedy algorithms:
   - **packing largest utility first**
     For this method objects are first sorted by their object utilities, and the most valuable objects are chosen first. The basis of this method is to pack the largest utilities first because they have more overall value than the other items, and fewer high-valued utilities can have a better overall value than a numerous low-valued utilities.
   - **packing smallest weight first**
     This packing method selects objects with the smallest weight first. We might pack more items if we first select those with the smallest weight, and more items might signify more overall value
   - **packing smallest utility first**
     This algorithm packs the objects with the smallest utility first. For example in a set of objects consisting of the utilities \(1,10,2,11,3,12,4\) we would pack item 1 first, item 2 second, item 3 third, and so on. The rationale behind this heuristic is that we might pack more items if we start with the smallest utilities, and more utilities can mean a larger overall value.
   - **packing largest weight first**
     The objects with the largest weight values are packed first. This method packs objects in the exact opposite order as packing objects with smallest weight values first. Some data sets have utilities that correspond to their weights (i.e.: larger utilities have larger weights). The idea behind this heuristic is to capitalize on these data sets and pack the utilities based on the weight constraint.

3. **Backtracking Methods**
   This method compares the current object to be packed with the next object, and unpacks the last object if next item cannot be packed. The reasoning behind this method is to pack more objects with a greater total utility than a simple greedy algorithm. When the last item is reached, it is unpacked in hopes of adding more...
than one item next in the list. The method sorts in two ways: sorted by largest object utility and sorted by smallest object utility. When the data is sorted by smallest object utility the next items in the list will have a higher (or equal) utility value, which may allow more items to be packed. When the data is sorted by largest object utility we might pack more valuable items initially before unpacking. The limitation to this method is that it does not repack an item if the next items do not yield a greater overall utility value.
Chapter 4: Implementation of Algorithms: Description of Java Code

We created a program (Appendix B) which implements the algorithms described above. The purpose of the program is to test the heuristics on each of the data sets and track the results for analysis. There are 21 data sets and seven packing algorithms. The program prompts the user for two inputs: first the test problem, and second the algorithm to pack items from the set. It describes the data set chosen by outputting the number of utilities, the optimal solution, and the maximum weight constraint. After each iteration of an algorithm, the output shows the utility packed, and the remaining space in the “knapsack.” The program stores these items in an array, and when no more items can be packed (see description of heuristics) outputs the items packed, the sum of the utilities, the optimal solution, and the percent difference between the experimental and optimal solutions.

The program was created over the course of 20 weeks using the Java environment BlueJ 2.2.0 developed by the University of Kent. We created three custom classes – Driver, pickItems, and sort which implements built-in java classes and our own methods and classes. The Java code used can be separated into three parts: Getting the test problem data, organizing the data, and packing certain items from the data.

Getting test problem data

In order to input the data from each test problem data set, the program uses a data file to store each problem. The user chooses which data set to analyze. Each data set is organized in the same way (see data file description), and is adaptable for varying problem sizes. Using a switch statement the program gets the user input regarding data set and algorithm to test. The program uses the Scanner class in order to obtain the data. Then it assigns variables such as the number of utilities (integers) and number of weights (integers) based on the data set file, and creates three arrays: the first holds the individual objects, the second is a one-dimensional array which holds the weights that correspond to each utility, and the third array contains the maximum sum value for the weights.

Organizing the data

After collecting the data from the data file, the program then sorts the data using the built in operator sort in the built-in array class. This class sorts data based on a parameter to be optimized. The program creates a copy of the constraints array and temporarily assigns it to another name while it sorts the data. The variables are sorted in the same order as their corresponding constraints.

Packing items

Now the program packs the data per selected algorithm. A public data member keeps track of the total space available (the constraint maximum) and subtracts the space of each object packed. When there is no more available space, the program stops adding objects to the “knapsack” and sums the total utility of the objects packed. Additionally the program outputs the optimal value and shows the percent difference with the experimental items chosen. Finally, the program outputs information about the items packed.
Additionally, we used a class called CombinationGenerator which finds all possible combinations of \( n \) objects in a set. We use a `for` loop to find the best possible combination by tracking the combinations where \( n \) goes from 0 to the number of items in the set. Our program uses `System.out` from the scanner class to output text in Notepad.

For debugging, we tested each method individually (as much as possible) by using hard-coded variables, and then changed them to flexible or final variables. After all the methods were created we used a variety of procedures in order to isolate certain functions or groups of code. For instance, we created data sets which were easy to solve by hand in order to test the accuracy of our program. The built-in debugger was very useful in stepping through the code and showing the value of each variable. Additionally, adding print statements within and after loops greatly aided in the debugging of the program.

The following shows the output after running data set 2 using a greedy algorithm which sorts by the largest object utility:

```
Enter the number of the test problem:
1: Multiple values, based on mknaps with first weight
2: Data set we created the first day
3: Data set where values are randomly chosen, no correspondence
4: Optimal solution found by taking smallest items first
5: Multiple values, based on mknaps with first weight
6: Multiple values, based on mknaps with first weight
7: Multiple values, based on mknaps with first weight
8: Optimal solution found by taking largest items first
9: Large Data set 1
10: Large Data set 2
11: Clumps of items: 4 clumps of similar-valued objects and clumped weights randomly generated
12: Data set where values are randomly chosen, no correspondence
13: Obj vals randomly generated, weights randomly generated
14: Optimal solution found by taking smallest items first, or smallest weights first
15: Optimal solution found by taking largest items first, or smallest weights first
16: Obj vals randomly generated, clumps of weights (in 5's and 10's)
17: 10 randomly generated items
18: 20 randomly generated items
19: 30 randomly generated items
20: 40 randomly generated items
21: 50 randomly generated items

Number of test problems: 1
Number of variables: 9
Number of constraints: 1
Optimal Solution: 2860

Enter a number:
1: Greedy Algorithm: Pick smallest weight first
2: Greedy Algorithm: Pick smallest value first
3: Greedy Algorithm: Pick largest weight first
4: Greedy Algorithm: Pick largest value first
5: Random Picker: Pick items randomly, using the random number generator
6: [not working] Look ahead method: compare the current object to be picked with the next object (sorted by smallest weight)
7: Look ahead method: compare the current object to be picked with the next object (sorted by smallest object value)
8: Look ahead method: compare the current object to be picked with the next object (sorted by largest object value)
9: Compute all possible combinations of objects and output best combinations

The maximum of weight is 1498
Pick the largest Object first

The item picked is: 2562
The remaining space is: 140

The item picked is: 160
The remaining space is: 44

The item picked is: 80
The remaining space is: 33

The item picked is: 40
The remaining space is: 9

The item picked is: 20
The remaining space is: 1

No more items will fit in the remaining space. We packed 5 items
The sum value of the items packed is: 2860
```
The optimal solution is 2860, which is a 0.0 percent difference.

We packed the following items: 2560 160 80 40 20

Would you like to select another data set and picking method? Y/N

The user can perform multiple analyses on different data sets by selecting to run the program again (by entering "Y") or end the program (by entering "N"). This is accomplished using a while loop to check the user’s input.

After running each algorithm on each data set, we tracked the results in a Microsoft Excel file (Table 1).
Chapter 5: Results

**Evaluation of algorithms**

We will evaluate packing methods based on the following factors:

1. Average performance: average difference from the optimal or best known solution
2. Closest to optimal solution: number of test sets for which this method was best

Using these to rank each packing method, we will perform a variety of analyses based on the following questions:

1. How often is a packing method ranked higher than another?
2. Are there certain algorithms that are consistently in the top three?
3. Are there certain algorithms that are consistently in the bottom three?

Further Questions:

1. How do the algorithms match up on certain data sets?
2. Are the data sets designed to unfairly weight some algorithms better or worse than others?

Tables 1 and 2 show the results for each heuristic on each data set. The name of the data file used is listed on the left side. The optimal solution for each data file is in the second column. When the optimal solution could not be found, the best value found is used as the “optimal.” These instances are denoted with a “....” The values in columns three, six, nine and twelve are the total utility of the objects packed using the data set (row) and particular algorithm (columns). For example, the solution found by using data set 15 and packing the smallest utility first is 105. Next to each algorithm column is a percent difference from optimal solution and a ranking column. The first shows the percent difference from the optimal solution and the second shows the “rank” of the algorithm in comparison with the other algorithms on the same data set. The method which comes closest to the optimal solution is ranked “1”, and the furthest away is ranked “9”. Methods that result in the same solution have the same rank (i.e.; more than one algorithm can be ranked 1 for a single data set).
Table 1: Summary of Results for each algorithm and data set

<table>
<thead>
<tr>
<th>DATA FILE</th>
<th>Optimal Solution</th>
<th>Pack largest utility first</th>
<th>Percent difference from optimal solution</th>
<th>Rank</th>
<th>Pack smallest utility first</th>
<th>Percent difference from optimal solution</th>
<th>Rank</th>
<th>Pack largest weight first</th>
<th>Percent difference from optimal solution</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_1.data</td>
<td>3900</td>
<td>3600</td>
<td>-7.69</td>
<td>1</td>
<td>2400</td>
<td>-38.46</td>
<td>5</td>
<td>3600</td>
<td>-7.69</td>
<td>1</td>
</tr>
<tr>
<td>data_2.data</td>
<td>98</td>
<td>75</td>
<td>-23.47</td>
<td>4</td>
<td>93</td>
<td>-5.10</td>
<td>2</td>
<td>58</td>
<td>-40.82</td>
<td>6</td>
</tr>
<tr>
<td>data_3.data</td>
<td>3190</td>
<td>2860</td>
<td>-10.34</td>
<td>2</td>
<td>1270</td>
<td>-60.19</td>
<td>5</td>
<td>2860</td>
<td>-10.34</td>
<td>2</td>
</tr>
<tr>
<td>data_4.data</td>
<td>78</td>
<td>78</td>
<td>0.00</td>
<td>1</td>
<td>30</td>
<td>-51.54</td>
<td>5</td>
<td>30</td>
<td>-61.54</td>
<td>5</td>
</tr>
<tr>
<td>data_5.data</td>
<td>4260</td>
<td>3670</td>
<td>-13.85</td>
<td>3</td>
<td>2235</td>
<td>-47.54</td>
<td>7</td>
<td>3285</td>
<td>-22.89</td>
<td>6</td>
</tr>
<tr>
<td>data_6.data</td>
<td>13245</td>
<td>13245</td>
<td>0.00</td>
<td>1</td>
<td>7445</td>
<td>-43.79</td>
<td>7</td>
<td>8890</td>
<td>-32.88</td>
<td>5</td>
</tr>
<tr>
<td>data_7.data</td>
<td>11498</td>
<td>8971</td>
<td>-21.98</td>
<td>4</td>
<td>8363</td>
<td>-27.27</td>
<td>6</td>
<td>8544</td>
<td>-25.69</td>
<td>5</td>
</tr>
<tr>
<td>data_8.data</td>
<td>3747</td>
<td>3747</td>
<td>0.00</td>
<td>1</td>
<td>3494</td>
<td>-6.75</td>
<td>5</td>
<td>3747</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>data_9.data</td>
<td>2557</td>
<td>2557</td>
<td>0.00</td>
<td>1</td>
<td>2360</td>
<td>-7.70</td>
<td>4</td>
<td>1612</td>
<td>-36.96</td>
<td>7</td>
</tr>
<tr>
<td>data_10.data</td>
<td>11688</td>
<td>11636</td>
<td>-0.44</td>
<td>2</td>
<td>7905</td>
<td>-32.37</td>
<td>6</td>
<td>6075</td>
<td>-48.02</td>
<td>7</td>
</tr>
<tr>
<td>data_11.data</td>
<td>14704</td>
<td>14704</td>
<td>0.00</td>
<td>1</td>
<td>11335</td>
<td>-22.91</td>
<td>6</td>
<td>14075</td>
<td>-4.28</td>
<td>3</td>
</tr>
<tr>
<td>data_12.data</td>
<td>549</td>
<td>549</td>
<td>0.00</td>
<td>1</td>
<td>417</td>
<td>-24.04</td>
<td>5</td>
<td>398</td>
<td>-27.50</td>
<td>6</td>
</tr>
<tr>
<td>data_13.data</td>
<td>13700</td>
<td>13700</td>
<td>0.00</td>
<td>1</td>
<td>1732</td>
<td>-87.36</td>
<td>6</td>
<td>5636</td>
<td>-58.86</td>
<td>4</td>
</tr>
<tr>
<td>data_14.data</td>
<td>300</td>
<td>80</td>
<td>-73.33</td>
<td>5</td>
<td>171</td>
<td>-43.00</td>
<td>3</td>
<td>69</td>
<td>-77.00</td>
<td>7</td>
</tr>
<tr>
<td>data_15.data</td>
<td>390</td>
<td>324</td>
<td>-16.92</td>
<td>2</td>
<td>105</td>
<td>-73.08</td>
<td>6</td>
<td>116</td>
<td>-70.26</td>
<td>4</td>
</tr>
<tr>
<td>data_16.data</td>
<td>6913</td>
<td>6854</td>
<td>-0.85</td>
<td>2</td>
<td>5255</td>
<td>-23.98</td>
<td>6</td>
<td>2623</td>
<td>-62.06</td>
<td>7</td>
</tr>
<tr>
<td>data_17.data</td>
<td>2543</td>
<td>2543</td>
<td>0.00</td>
<td>1</td>
<td>1532</td>
<td>-39.76</td>
<td>5</td>
<td>1310</td>
<td>-48.49</td>
<td>6</td>
</tr>
<tr>
<td>data_18.data</td>
<td>4571</td>
<td>4337</td>
<td>-5.12</td>
<td>2</td>
<td>3448</td>
<td>-24.57</td>
<td>3</td>
<td>2551</td>
<td>-44.19</td>
<td>7</td>
</tr>
<tr>
<td>data_19.data</td>
<td>8644</td>
<td>8644</td>
<td>0.00</td>
<td>1</td>
<td>7099</td>
<td>-17.87</td>
<td>4</td>
<td>5507</td>
<td>-36.29</td>
<td>7</td>
</tr>
<tr>
<td>data_20.data</td>
<td>8445</td>
<td>8445</td>
<td>0.00</td>
<td>1</td>
<td>5762</td>
<td>-31.77</td>
<td>6</td>
<td>5070</td>
<td>-39.96</td>
<td>7</td>
</tr>
<tr>
<td>data_21.data</td>
<td>7874</td>
<td>1286</td>
<td>-83.67</td>
<td>3</td>
<td>521</td>
<td>-93.38</td>
<td>6</td>
<td>369</td>
<td>-95.31</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATA FILE</th>
<th>Optimal Solution</th>
<th>Pack smallest weight first</th>
<th>Percent difference from optimal solution</th>
<th>Rank</th>
<th>(sort smallest utility) Backtrack 1</th>
<th>Percent difference from optimal solution</th>
<th>Rank</th>
<th>(sort largest utility) Backtrack 2</th>
<th>Percent difference from optimal solution</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_1.data</td>
<td>3900</td>
<td>2400</td>
<td>-38.46</td>
<td>5</td>
<td>3600</td>
<td>-7.69</td>
<td>1</td>
<td>2500</td>
<td>-35.90</td>
<td>4</td>
</tr>
<tr>
<td>data_2.data</td>
<td>98</td>
<td>91</td>
<td>-7.14</td>
<td>3</td>
<td>98</td>
<td>0.00</td>
<td>1</td>
<td>58</td>
<td>-40.82</td>
<td>6</td>
</tr>
<tr>
<td>data_3.data</td>
<td>3190</td>
<td>1270</td>
<td>-60.19</td>
<td>5</td>
<td>3190</td>
<td>0.00</td>
<td>1</td>
<td>2230</td>
<td>-30.09</td>
<td>4</td>
</tr>
<tr>
<td>data_4.data</td>
<td>78</td>
<td>78</td>
<td>0.00</td>
<td>1</td>
<td>68</td>
<td>-12.82</td>
<td>3</td>
<td>68</td>
<td>-12.82</td>
<td>3</td>
</tr>
<tr>
<td>data_5.data</td>
<td>4260</td>
<td>3465</td>
<td>-18.66</td>
<td>4</td>
<td>3715</td>
<td>-12.79</td>
<td>2</td>
<td>3885</td>
<td>-8.80</td>
<td>1</td>
</tr>
<tr>
<td>data_6.data</td>
<td>13245</td>
<td>9905</td>
<td>-25.22</td>
<td>3</td>
<td>9595</td>
<td>-27.56</td>
<td>4</td>
<td>12790</td>
<td>-3.51</td>
<td>2</td>
</tr>
<tr>
<td>data_7.data</td>
<td>11498</td>
<td>7312</td>
<td>-36.41</td>
<td>7</td>
<td>11498</td>
<td>0.00</td>
<td>1</td>
<td>9701</td>
<td>-15.63</td>
<td>3</td>
</tr>
<tr>
<td>data_8.data</td>
<td>3747</td>
<td>3494</td>
<td>-6.75</td>
<td>5</td>
<td>3496</td>
<td>-6.70</td>
<td>4</td>
<td>3694</td>
<td>-1.41</td>
<td>3</td>
</tr>
<tr>
<td>data_9.data</td>
<td>2557</td>
<td>2241</td>
<td>-12.36</td>
<td>6</td>
<td>2361</td>
<td>-7.67</td>
<td>3</td>
<td>2586</td>
<td>-0.04</td>
<td>2</td>
</tr>
<tr>
<td>data_10.data</td>
<td>11688</td>
<td>10733</td>
<td>-8.17</td>
<td>3</td>
<td>8038</td>
<td>-31.23</td>
<td>5</td>
<td>11688</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>data_11.data</td>
<td>14704</td>
<td>13388</td>
<td>-9.09</td>
<td>4</td>
<td>11338</td>
<td>-22.89</td>
<td>5</td>
<td>14699</td>
<td>-0.03</td>
<td>2</td>
</tr>
<tr>
<td>data_12.data</td>
<td>549</td>
<td>444</td>
<td>-19.13</td>
<td>3</td>
<td>419</td>
<td>-23.68</td>
<td>4</td>
<td>537</td>
<td>-2.19</td>
<td>2</td>
</tr>
<tr>
<td>data_13.data</td>
<td>13700</td>
<td>10454</td>
<td>-23.69</td>
<td>3</td>
<td>5184</td>
<td>-62.16</td>
<td>5</td>
<td>13616</td>
<td>-0.61</td>
<td>2</td>
</tr>
<tr>
<td>data_14.data</td>
<td>300</td>
<td>105</td>
<td>-65.00</td>
<td>4</td>
<td>178</td>
<td>-40.67</td>
<td>2</td>
<td>300</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>data_15.data</td>
<td>390</td>
<td>390</td>
<td>0.00</td>
<td>1</td>
<td>116</td>
<td>-70.26</td>
<td>4</td>
<td>323</td>
<td>-17.18</td>
<td>3</td>
</tr>
<tr>
<td>data_16.data</td>
<td>6913</td>
<td>6597</td>
<td>-4.57</td>
<td>4</td>
<td>6115</td>
<td>-11.54</td>
<td>5</td>
<td>6754</td>
<td>-2.30</td>
<td>3</td>
</tr>
<tr>
<td>data_17.data</td>
<td>2543</td>
<td>2543</td>
<td>0.00</td>
<td>1</td>
<td>1226</td>
<td>-51.79</td>
<td>7</td>
<td>2399</td>
<td>-5.66</td>
<td>4</td>
</tr>
<tr>
<td>data_18.data</td>
<td>4571</td>
<td>3236</td>
<td>-29.21</td>
<td>4</td>
<td>3135</td>
<td>-31.42</td>
<td>5</td>
<td>4375</td>
<td>-4.29</td>
<td>1</td>
</tr>
<tr>
<td>data_19.data</td>
<td>8644</td>
<td>8321</td>
<td>-3.74</td>
<td>3</td>
<td>6674</td>
<td>-22.79</td>
<td>6</td>
<td>8520</td>
<td>-1.43</td>
<td>2</td>
</tr>
<tr>
<td>data_20.data</td>
<td>8445</td>
<td>8063</td>
<td>-4.52</td>
<td>3</td>
<td>6238</td>
<td>-26.13</td>
<td>4</td>
<td>8322</td>
<td>-1.46</td>
<td>2</td>
</tr>
<tr>
<td>data_21.data</td>
<td>7874</td>
<td>1164</td>
<td>-85.22</td>
<td>4</td>
<td>7597</td>
<td>-3.52</td>
<td>2</td>
<td>7874</td>
<td>0.00</td>
<td>1</td>
</tr>
</tbody>
</table>
### Table 2: Summary of findings regarding different algorithms

<table>
<thead>
<tr>
<th>DATA FILE</th>
<th>Optimal Solution</th>
<th>Pack Randomity</th>
<th>Percent difference from optimal solution</th>
<th>Rank</th>
<th>All Combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_1</td>
<td>3900</td>
<td>2400</td>
<td>-38.46</td>
<td>5</td>
<td>3900</td>
</tr>
<tr>
<td>data_2</td>
<td>98</td>
<td>60</td>
<td>-36.78</td>
<td>5</td>
<td>95</td>
</tr>
<tr>
<td>data_3</td>
<td>3190</td>
<td>1270</td>
<td>-60.19</td>
<td>5</td>
<td>2860</td>
</tr>
<tr>
<td>data_4</td>
<td>78</td>
<td>30</td>
<td>-61.54</td>
<td>5</td>
<td>78</td>
</tr>
<tr>
<td>data_5</td>
<td>4260</td>
<td>3465</td>
<td>-18.66</td>
<td>4</td>
<td>4260</td>
</tr>
<tr>
<td>data_6</td>
<td>113245</td>
<td>7925</td>
<td>-40.17</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>data_7</td>
<td>11498</td>
<td>10031</td>
<td>-12.76</td>
<td>2</td>
<td>10618</td>
</tr>
<tr>
<td>data_8</td>
<td>3747</td>
<td>3494</td>
<td>-6.75</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>data_9</td>
<td>11488</td>
<td>9774</td>
<td>-16.38</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>data_10</td>
<td>14704</td>
<td>11335</td>
<td>-22.91</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>data_11</td>
<td>549</td>
<td>374</td>
<td>-31.88</td>
<td>7</td>
<td>544</td>
</tr>
<tr>
<td>data_12</td>
<td>13700</td>
<td>1730</td>
<td>-87.37</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>data_13</td>
<td>300</td>
<td>80</td>
<td>-73.33</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>data_14</td>
<td>390</td>
<td>105</td>
<td>-73.08</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>data_15</td>
<td>6913</td>
<td>6913</td>
<td>0.00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>data_16</td>
<td>2543</td>
<td>2437</td>
<td>-4.17</td>
<td>3</td>
<td>2543</td>
</tr>
<tr>
<td>data_17</td>
<td>4571</td>
<td>3056</td>
<td>-33.14</td>
<td>6</td>
<td>4571</td>
</tr>
<tr>
<td>data_18</td>
<td>8644</td>
<td>6856</td>
<td>-20.68</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>data_19</td>
<td>8445</td>
<td>5904</td>
<td>-30.09</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>data_20</td>
<td>7874</td>
<td>1071</td>
<td>-86.40</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Average Percent difference from Optimal Solution**

<table>
<thead>
<tr>
<th></th>
<th>Pack largest utility first</th>
<th>Pack smallest utility first</th>
<th>(sort largest value)</th>
<th>(sort smallest value)</th>
<th>Pack largest weight first</th>
<th>Pack smallest weight first</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.27</td>
<td>8.77</td>
<td>22.54</td>
<td>21.79</td>
<td>36.46</td>
<td>38.69</td>
<td>40.53</td>
</tr>
</tbody>
</table>

**Number of times ranked**

<table>
<thead>
<tr>
<th></th>
<th>Pack largest utility first</th>
<th>Pack smallest utility first</th>
<th>(sort largest value)</th>
<th>(sort smallest value)</th>
<th>Pack largest weight first</th>
<th>Pack smallest weight first</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of times ranked 1</td>
<td>11</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of times ranked 1, 2, or 3</td>
<td>18</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of times ranked 2, or 3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Average Rank</td>
<td>1.90</td>
<td>2.52</td>
<td>3.52</td>
</tr>
</tbody>
</table>

Table 2 summarizes the results for each packing algorithm with methods organized by their average rankings. This table is helpful because it shows which algorithms were consistently closest to the optimal utility.

**Ranking and average percent difference**

We created a random packing algorithm to test against the other packing methods. According to Table 2, the random packer has an average rank of 4.90 with four packing methods averaging better and two methods averaging worse at approximating the optimal solution. Chart 2 shows the difference in approximation for each data set for the random packer. We found that the random packer faired decently well against the optimal
solution. However, it nearly always underestimated the optimal solution and only ranked first against the other algorithms once (data set 16).

We expected these results because it makes sense to choose the most valuable objects first (i.e.; choosing the most valuable objects first, or the smallest weights first). This also shows that packing the largest weight first and packing the smallest utility first are poor heuristics, because they were worse than the random packer.

*Chart 2: The optimal solution and the random packer*

![Chart 2: The optimal solution and the random packer](image)

It seems like the average percent difference from the optimal solution (Chart 4) should correspond to the ranking, however this is not the case. *Backtrack 2* is ranked overall 2\textsuperscript{nd} and averaged a 8.77 percent difference from the optimal solution, and *packing the largest utility first* is ranked overall 1\textsuperscript{st} and averaged 12.27 percent different. This happens because rankings do not accurately reflect small differences in results whereas the average percent difference does. *Backtrack 2* did not rank closest to the optimal solution more times than *packing the largest utility first*, but on average is closer to the optimal solution. Also *packing the largest utility first* performed very poorly on data sets 14 and 21 which skews the results by enlarging the average percent difference. Taking both of these analyses into account is better than using just one method of evaluation. Since the goal is to find the objects to pack which maximize the utility subject to the weight constraint, the average percent difference is a better metric with which to rank the methods. Charts 3 and 4 show this difference:
The following chart shows how many times each of the heuristics were ranked best, in the top three, or in the worst three. It’s a good indication that an algorithm approximates the optimal solution if it has high numbers of best and top three and few numbers of worst three. From the following chart we see that Backtrack2 and packing the largest utility first are in the top three 18 and 17 times respectively, and ranked first the greatest number of times. It is possible that the number of data sets best packed by choosing the largest utility first is higher than the other types of data sets and this contributes to the difference between such methods. It is notable that packing the largest weight and packing the smallest weight are best a similar number of times (2, 3
respectively) however packing the largest weight is in the worst three 8 times whereas packing the smallest constraint is only in the worst three rankings once. This shows that it is important to look at more than one kind of ranking when analyzing the effectiveness of an algorithm. By looking at three different types of ranking data we see that packing the smallest weight first is nearly always better than packing the largest weight first which follows intuition.

*Chart 5: Number of times each heuristic was ranked Best, Top Three, Worst Three*

**Greedy algorithms and backtracking methods compared to optimal solution**

It is also important to identify which algorithms are consistently closest to the optimal solutions and which are not. Packing by largest utility, backtrack1, backtrack2, and packing the smallest weight are all consistently closest to the optimal solution whereas the others are not in the top three methods closest to the optimal solution. The following chart shows the difference between the backtracking methods and the optimal solution:
Chart 6: Optimal Solution and Backtracking Algorithms

Backtrack 2 fairs better than Backtrack 1 on most data sets (5, 6, 8-21) but Backtrack 1 is within 25% of the optimal solution in its approximations for 13 of the 21 data sets. Chart 7 shows the results for each data set for the greedy algorithms:

Chart 7: Optimal Solution & Greedy Algorithms
We specifically designed data sets to work best with certain algorithms, such as an algorithm which packs the largest utilities first. How do other heuristics match-up on these data sets?

Clumped Data

We created some data sets with clumps of data—either clumps of utilities, weights or both. We tested each algorithm on these data sets and the following charts compare the performance of each heuristic on the data sets of clumped values. Each chart is sorted in the same order.

Chart 8: Data set 11 (clumps of utilities and clumps of weights)

Chart 9: Data set 13 (clumps of utilities and random weights)
Chart 10: Data set 16 (clumps of weights and consecutive utilities)

There is no clear pattern as to how these heuristics perform on the clumped data. Data set 11 has somewhat equal results for each heuristic while data sets 13 and 16 have similar patterns such that packing the largest weight first has the worst results. However, there are some similarities between each of the three charts: Packing the largest utility first and packing the largest weight first are closest to the optimal solution for these clumped data sets. Also, data sets 11, 13, and 16 follow a similar pattern (with minor anomalies in data sets 13 and 16) such that the algorithms are in a descending order and in general, the same algorithms perform better than others on the clumped data sets.

Large data sets

We created some large data sets (greater than 25 items) and tested each algorithm on them. The following charts show a general trend such that Backtrack 2 and packing the largest utility first are closest to the optimal solution and the other heuristics tend to rank in the same order for the sets shown.

Chart 11: Data set 20
Chart 12: Data set 19

Chart 13: Data set 16

Chart 14: Data set 10

Chart 15: Data set 6
The following large data sets show a different trend. The performance of the heuristics is relatively equal and the spread between the best and worst results is not as great as the other data sets.

**Chart 16: Data set 9**

![Chart 16: Data set 9](image)

**Chart 17: Data set 11**

![Chart 17: Data set 11](image)

**Chart 18: Data set 7**

![Chart 18: Data set 7](image)

Data sets 15 and 13, and follow yet another trend. These amplify the differences shown in the first trend. *Backtrack1, pack smallest utility first, pack randomly* and *pack largest weight first* are much lower than the other heuristics. Data set 15 has an optimal solution which is found best by taking the smallest utility first and data set 13 are clumps of utilities with random weights.
The following charts do not follow any of the aforementioned trends, but have interesting trends as an entity.

Backtrack2 performs best on data set 14 (like most other sets) and the other heuristics are progressively worse approximations of the optimal solution. All of the charts are ordered the same, and no other chart demonstrates similar behavior.
Both backtracking methods perform very well on this data set but all of the other methods fail horribly at approximating the optimal solution. No other data set has similar trends in performance.

**Small data sets**

The small data sets (1, 2, 3, 4, 5, 8, 12, 17, 18) have very few trends, unlike the large data sets. Perhaps this is because the small data sets are less random in nature (i.e.; created to work best or fail with certain heuristics). Nonetheless, data sets 18, 12, and 4 have similar results with each heuristic. In general Backtrack2 is the best approximation, and the rest of the heuristics are perform progressively worse.

**Chart 23: data set 4**

![Chart 23: data set 4](image)

**Chart 24: data set 12**

![Chart 24: data set 12](image)

**Chart 25: data set 18**

![Chart 25: data set 18](image)

The other data sets do not follow any trend and it is difficult to find any generalizations. Some data sets have relatively equal results, such as data set 5 shown in chart 26. In general backtrace2 performs best on the small data sets. Some sets do not
follow this trend, such as data set 2 (chart 27). Other sets have opposite trends, such as data set 2 and data set 17 (see charts 27 and 28).

*Chart 26: data set 5*

*Chart 27: data set 2*

*Chart 28: data set 17*

Without generating more small data sets it is hard to generalize any new trends regarding characteristics of small data sets. In a real life setting small data sets will have solutions that are easily found without the use of heuristics, however we can attempt to characterize small data sets in other useful ways so that we can apply such techniques to larger data sets.
Questions to consider

1. Do we have enough data sets to generate accurate and useful analyses?

   It is always a good idea to have more data sets to test heuristics, but we can still make generalizations based on the number and type of data sets and algorithms we have. We know that the backtracking method seems to work consistently well on many types of data sets, and more forms of backtracking will probably be better than a random packing method or a greedy algorithm that sorts by largest weight first.

2. Is it possible that certain packing methods work best on different data sets?

   Some data sets were made to work best with certain packing algorithms, however other packing methods come relatively close to approximating the optimal solution too. We can use this information for different applications: if we know that a data set has a certain characteristic then we can optimize our algorithm such that it incorporates the characteristics that we know. For instance, if we know that the objects in a set have weights corresponding to the utility we can capitalize on this knowledge by choosing a heuristic that incorporates this knowledge. If we do not know anything about a data set, we can select a method which is consistently close to the optimal solution of many types of data sets using evaluation techniques such as the percent difference average.

3. Is it acceptable to conclude that an algorithm works best on one type of data set, but fails at all other types of sets?

   It is acceptable to conclude that an algorithm works best on a certain type of data set, and it is equally important to use this knowledge appropriately. It is difficult to show that an algorithm always works best in comparison to other methods, but with trial and error, and well thought-out experiments such knowledge can be invaluable. For example, if we know that the price a cow sells at is always directly related to its weight we can incorporate this into a heuristic which selects which cows to sell.

4. Is it more feasible to sort and pack from large data sets?

   We use large data sets because they are realistic numbers that mimic real life situations. Solutions for small data sets are easier to find even without the use of a super computer, and are less feasible representations of actual applications of heuristics. However, it is realistic to practice on small data sets to mimic behaviors with large data sets because it allows us to check accuracy of the packing methods.

5. Do the data sets we have accurately reflect real-life data? What are some applications or examples of my data sets in a real-life setting?
We have three main types of data sets: Clumped data, small data sets, and large data sets. Each of these can accurately reflect real-life data. For example, clumped data can represent the age and weight of cattle; calves weigh less than steers, but often calves weigh the same in the first year, and steers sold at auctions are of a similar weight. Data sets where the utilities and weights correspond can be seen at the post office. The more a package weighs, the more it costs to ship it. Also the distance a package travels corresponds the cost of shipping. Another example that requires careful consideration is flat rate packages at the post office. Such packages cost the same regardless of weight, but must be contained to a box of the same dimensions. The packages shipped through the US post office can represent large data sets without any trends, but a shipping company specializing in moving one type of item, like produce, usually has characteristics which can help in the creation of a heuristic used with the large data set.
Chapter 5: Conclusions and Future Work

We explored the knapsack problem using a variety of basic heuristics and 21 data sets. After tracking the results of each heuristic and comparing trends, we found that packing largest value first was best most often but backtracking1 on average approximates the optimal solution with a smaller error.

Certain types of data sets work best with certain types of heuristics and this is valuable information for industrial applications of heuristics for situations in which we know characteristics of the data set. For example, if we know that larger utilities correspond to smaller weights, then we would first pack items by largest utility value.

For data sets with over 20 items we were unable to find the optimal solution due to the constraints of our available technology. With access to a super computer and time to run our heuristics on multiple data sets we might have more information to compare our results. To mitigate this issue we simply used the best approximation available as the optimal solution.

With more time we would further explore these heuristics, expand our ideas to include more complex algorithms, and make use of a larger variety of test data sets. We would investigate lookahead heuristics and more combinations with our current packing methods. For example, we might create a backtracking method that unpacks 5 items instead of 1 like in backtrack1 and backtrack2. Additionally we could use tabu search methods to approximate the optimal solution. Also we would explore other greedy algorithms such as packing items with the greatest utility/weight ratio. It is possible that the number and type of data sets we used skewed the results because backtracking might work better on the limited data sets we have and worse on other types. In the future we would design an equal number of different types of data sets for a more balanced analysis.

Another issue is the feasibility of using just one constraint value per utility. Many real-life problems are multidimensional. In the future we would include algorithms which take into account more than one weight, and these results could significantly impact our current findings.

Heuristics are important methods for the knapsack problem because they can greatly reduce the time needed to "pack" items or choose items to pack into a container. These sorts of problems include many real-life situations which might not seem like a knapsack problem, such as the cow-milking example mentioned. With effective use heuristics we can save money and time for producers and consumers in industrial and other settings.
References


Appendix

Appendix A: Data sets used in analysis

data_1.data
7
 6 1 3900
 100 600 1200 2400 500 2000
 8 12 13 64 22 41
80

data_2.data
1
 6 1 95
 30 25 23 5 20 20
 50 20 20 15 10 10
 90

data_3.data
1
 9 1 2860
 10 20 40 80 160 320 640 1280 2560
 9 8 24 11 96 305 489 641 1358
1498

data_4.data
1
 1 4 1 78
 10 20 30 28
 100 90 70 20
200

data_5.data
1
 15 1 4260
 100 220 90 400 300 400 205 120 160 580 400 140 100 1300 650
 8 24 13 80 70 80 45 15 28 90 130 32 20 120 40
550

data_6.data
7
 28 1 12400
 100 220 90 400 300 400 205 120 160 580 400 140 100 1300 650 320 480 80 60 2550
 3100 1100 950 450 300 220 200 520
8 24 13 80 70 80 45 15 28 90 130 32 20 120 40 30 20 6 3 180
220 50 30 50 12 5 8 10
930

data_7.data
7
 39 1 10618
 560 1125 300 620 2100 431 68 328 47 122 332 196 41 25 425 4260 416 115 82 22 631
 132 420 86 42 103 215 81 91 26 49 420 316 72 71 49 108 116 90
 40 91 10 10 160 20 3 12 3 18 9 25 1 1 10 280 10 8 1 1 49
 8 21 6 1 5 10 8 2 1 0 10 42 6 4 8 0 10 1
600

data_8.data
1
 10 1 3700
 54 98 99 100 250 600 700 795 798 800
 216 392 396 400 1000 2400 2800 3180 3192 3200
15000
**data_9.data**

```
 1 253 1 2000
11 13 10 11 16 18 1 16 1 17 16
20 13 16 10 11 8 9 6 15 7 15
14 11 7 16 3 19 19 12 8 12
15 14 4 6 11 16 8 3 1 9
5 14 12 5 19 18 4 10 16 4 13
1 17 7 8 17 14 11 6 9 10 5
19 17 20 5 11 16 9 11 3 1 19
0 15 1 4 13 14 15 12 0 6 16
12 4 6 5 19 20 6 15 2 16 4
15 1 16 14 3 8 17 6 2 11 6
18 8 17 12 4 11 4 2 8 20 10
4 3 13 0 3 13 6 18 8 13 3
9 10 9 18 3 1 11 13 1 3 4
17 1 16 1 15 4 18 14 15 9 5
19 16 19 18 4 14 19 15 9 19 7
9 13 11 18 8 5 5 5 4 19 10 20
12 5 19 2 13 9 13 5 3 13 3
7 11 7 20 20 2 18 18 5 13 11
6 16 6 9 5 5 13 13 8 10 2 3
7 11 5 18 13 12 6 15 2 16
7 19 15 0 2 4 1 12 12 14 14
14 5 5 10 12 8 9 8 2 20 11
18 3 16 14 20 8 7 2 10 9 8
91 92 237 154 169 189 187 168 183 120 225
12 139 102 229 176 108 62 43 192 117 56
135 145 76 155 124 110 65 129 177 164 31
147 131 47 151 191 213 215 161 94 33 147
161 51 204 60 17 156 225 137 83 138 148
50 207 228 208 41 118 77 112 157 52 90
116 64 42 132 10 177 180 75 54 111 3
198 21 16 98 62 0 46 115 75 162 173
205 162 248 101 105 121 193 100 105 116 35
176 72 69 194 196 168 142 154 112 229 240
167 181 197 128 111 114 240 147 55 93 68
69 120 176 163 122 138 172 221 157 30 155
65 140 247 158 184 179 61 231 209 189 45
219 176 0 219 51 156 211 54 32 245 162
152 92 11 116 114 202 162 144 180 148 82
233 128 108 233 83 182 2 149 86 244 173
29 212 100 220 130 10 167 186 206 153 17
167 157 138 172 155 87 117 160 161 233 126
122 24 134 232 160 75 203 44 142 43 193
235 173 139 174 4 50 236 169 233 67 83
5 130 75 238 92 142 221 158 216 170 25
203 184 90 49 96 53 169 15 211 106 5
172 237 99 83 134 62 198 174 237 28 27
```

**data_10.data**

```
 1 255 1 12000
13 85 97 58 0 46 76 95 22 11 58
21 41 40 6 13 69 95 9 14 40
50 51 71 86 72 19 13 7 92 57 18
42 47 92 62 51 85 21 8 57 36 85
2 74 20 97 3 60 25 88 11 40 49
67 88 72 73 79 0 82 83 21 50 93
26 13 64 32 23 77 10 97 21 55 48
72 76 24 37 81 55 42 55 64 24 64
78 0 57 97 63 0 25 84 73 36 52
53 91 25 93 26 0 4 57 46 17 14
46 27 62 80 20 39 32 90 80 84 17
85 14 10 86 25 65 44 51 37 84
20 96 7 25 86 85 74 87 24 21 85
```
<table>
<thead>
<tr>
<th></th>
<th>35</th>
<th>38</th>
<th>75</th>
<th>4</th>
<th>20</th>
<th>78</th>
<th>81</th>
<th>69</th>
<th>9</th>
<th>64</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>93</td>
<td>73</td>
<td>26</td>
<td>67</td>
<td>58</td>
<td>29</td>
<td>32</td>
<td>15</td>
<td>71</td>
<td>17</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>63</td>
<td>73</td>
<td>39</td>
<td>4</td>
<td>85</td>
<td>67</td>
<td>87</td>
<td>27</td>
<td>51</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>55</td>
<td>43</td>
<td>24</td>
<td>23</td>
<td>57</td>
<td>83</td>
<td>64</td>
<td>77</td>
<td>90</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>45</td>
<td>88</td>
<td>59</td>
<td>23</td>
<td>5</td>
<td>38</td>
<td>83</td>
<td>36</td>
<td>53</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>47</td>
<td>39</td>
<td>21</td>
<td>69</td>
<td>75</td>
<td>21</td>
<td>70</td>
<td>92</td>
<td>69</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>81</td>
<td>30</td>
<td>15</td>
<td>23</td>
<td>42</td>
<td>32</td>
<td>52</td>
<td>18</td>
<td>13</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>41</td>
<td>97</td>
<td>30</td>
<td>95</td>
<td>99</td>
<td>62</td>
<td>74</td>
<td>93</td>
<td>62</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>81</td>
<td>43</td>
<td>61</td>
<td>87</td>
<td>81</td>
<td>66</td>
<td>8</td>
<td>41</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>6</td>
<td>57</td>
<td>44</td>
<td>93</td>
<td>83</td>
<td>1</td>
<td>36</td>
<td>40</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td>49</td>
<td>93</td>
<td>40</td>
<td>95</td>
<td>92</td>
<td>17</td>
<td>66</td>
<td>87</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>92</td>
<td>13</td>
<td>74</td>
<td>67</td>
<td>83</td>
<td>31</td>
<td>77</td>
<td>87</td>
<td>17</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>81</td>
<td>40</td>
<td>15</td>
<td>47</td>
<td>22</td>
<td>88</td>
<td>84</td>
<td>60</td>
<td>19</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>42</td>
<td>57</td>
<td>30</td>
<td>75</td>
<td>66</td>
<td>0</td>
<td>45</td>
<td>85</td>
<td>85</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>73</td>
<td>30</td>
<td>70</td>
<td>11</td>
<td>84</td>
<td>39</td>
<td>43</td>
<td>98</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>76</td>
<td>17</td>
<td>15</td>
<td>14</td>
<td>4</td>
<td>97</td>
<td>23</td>
<td>100</td>
<td>18</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>67</td>
<td>28</td>
<td>70</td>
<td>90</td>
<td>16</td>
<td>46</td>
<td>34</td>
<td>68</td>
<td>87</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>42</td>
<td>0</td>
<td>11</td>
<td>39</td>
<td>42</td>
<td>65</td>
<td>57</td>
<td>52</td>
<td>38</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>59</td>
<td>58</td>
<td>86</td>
<td>76</td>
<td>97</td>
<td>64</td>
<td>83</td>
<td>31</td>
<td>28</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>22</td>
<td>28</td>
<td>30</td>
<td>53</td>
<td>38</td>
<td>14</td>
<td>7</td>
<td>93</td>
<td>41</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>89</td>
<td>68</td>
<td>61</td>
<td>82</td>
<td>92</td>
<td>63</td>
<td>93</td>
<td>14</td>
<td>97</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>45</td>
<td>69</td>
<td>50</td>
<td>37</td>
<td>3</td>
<td>34</td>
<td>96</td>
<td>37</td>
<td>33</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>31</td>
<td>32</td>
<td>13</td>
<td>7</td>
<td>31</td>
<td>50</td>
<td>53</td>
<td>44</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>20</td>
<td>48</td>
<td>11</td>
<td>15</td>
<td>62</td>
<td>28</td>
<td>53</td>
<td>65</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>50</td>
<td>50</td>
<td>99</td>
<td>42</td>
<td>76</td>
<td>88</td>
<td>70</td>
<td>61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>96</td>
<td>60</td>
<td>7</td>
<td>83</td>
<td>86</td>
<td>76</td>
<td>25</td>
<td>53</td>
<td>31</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>96</td>
<td>18</td>
<td>61</td>
<td>67</td>
<td>38</td>
<td>45</td>
<td>20</td>
<td>21</td>
<td>62</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>31</td>
<td>42</td>
<td>68</td>
<td>0</td>
<td>49</td>
<td>86</td>
<td>79</td>
<td>58</td>
<td>71</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>68</td>
<td>85</td>
<td>52</td>
<td>2</td>
<td>47</td>
<td>37</td>
<td>51</td>
<td>22</td>
<td>15</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>29</td>
<td>58</td>
<td>61</td>
<td>96</td>
<td>81</td>
<td>99</td>
<td>54</td>
<td>25</td>
<td>56</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>76</td>
<td>42</td>
<td>74</td>
<td>55</td>
<td>33</td>
<td>54</td>
<td>37</td>
<td>46</td>
<td>42</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>36</td>
<td>25</td>
<td>25</td>
<td>48</td>
<td>82</td>
<td>98</td>
<td>55</td>
<td>35</td>
<td>55</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>92</td>
<td>27</td>
<td>19</td>
<td>56</td>
<td>53</td>
<td>65</td>
<td>33</td>
<td>47</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10000

data_11.data
1
40 1 14000
10 11 12 12 14 15 16 17 17 18 100
100 100 101 103 104 105 107 107 107 500 500
501 503 505 505 506 508 508 850 852 853
853 854 856 856 856 857 859
60 61 61 60 60 61 61 60 61 61 6
7 7 6 6 7 6 7 7 7 101 101
101 100 101 100 100 101 101 86 85 86
85 86 86 86 86 85 85
2250

data_12.data
1
13 1 535
7 68 55 38 70 5 12 44 47 69 45 68
59 64 31 16 33 27 39 78 17 36 48 28
400

data_13.data
1
40 1 14000
10 12 13 13 14 15 16 18 20 21 100
101 103 103 105 106 107 107 108 500 501
502 502 503 505 505 507 508 850 851 851
852 854 855 856 857 857 858
62 105 37 53 120 12 101 104 103 92 93
88 119 114 129 136 79 33 140 13 104 37
66 32 68 60 132 109 119 113 32 55 38
111 14 107 5 52 127 91
1500

- 31 -
<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>20000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td>5000</td>
<td>2500</td>
<td>1250</td>
<td>625</td>
<td>313</td>
<td>156</td>
<td>78</td>
<td>39</td>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**data_15.data**

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>1400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>24</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td>5000</td>
<td>2500</td>
<td>1250</td>
<td>625</td>
<td>313</td>
<td>156</td>
<td>78</td>
<td>39</td>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**data_16.data**

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75</td>
<td>16800</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>114</td>
<td>46</td>
<td>123</td>
<td>123</td>
<td>93</td>
<td>139</td>
<td>6</td>
<td>178</td>
<td>137</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>26</td>
<td>141</td>
<td>152</td>
<td>140</td>
<td>179</td>
<td>104</td>
<td>111</td>
<td>69</td>
<td>153</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>26</td>
<td>21</td>
<td>166</td>
<td>17</td>
<td>120</td>
<td>74</td>
<td>151</td>
<td>143</td>
<td>79</td>
<td>159</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>170</td>
<td>130</td>
<td>127</td>
<td>59</td>
<td>51</td>
<td>22</td>
<td>43</td>
<td>72</td>
<td>106</td>
<td>169</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>32</td>
<td>20</td>
<td>108</td>
<td>171</td>
<td>167</td>
<td>140</td>
<td>122</td>
<td>76</td>
<td>86</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>57</td>
<td>76</td>
<td>44</td>
<td>43</td>
<td>38</td>
<td>154</td>
<td>78</td>
<td>28</td>
<td>15</td>
<td>15</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>105</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>8000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**data_17.data**

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>2615</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>416</td>
<td>258</td>
<td>464</td>
<td>450</td>
<td>102</td>
<td>169</td>
<td>208</td>
<td>236</td>
<td>25</td>
<td>342</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>28</td>
<td>29</td>
<td>10</td>
<td>49</td>
<td>35</td>
<td>37</td>
<td>15</td>
<td>42</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**data_18.data**

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>1</td>
<td>5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>446</td>
<td>261</td>
<td>72</td>
<td>281</td>
<td>15</td>
<td>294</td>
<td>235</td>
<td>444</td>
<td>293</td>
<td>224</td>
<td>491</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>178</td>
<td>108</td>
<td>411</td>
<td>392</td>
<td>3</td>
<td>300</td>
<td>312</td>
<td>68</td>
<td>412</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>16</td>
<td>16</td>
<td>26</td>
<td>12</td>
<td>30</td>
<td>34</td>
<td>39</td>
<td>43</td>
<td>3</td>
<td>39</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>31</td>
<td>19</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>16</td>
<td>5</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>2500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**data_19.data**

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>1</td>
<td>8880</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>486</td>
<td>427</td>
<td>76</td>
<td>211</td>
<td>162</td>
<td>459</td>
<td>280</td>
<td>236</td>
<td>195</td>
<td>486</td>
<td>316</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>271</td>
<td>412</td>
<td>316</td>
<td>456</td>
<td>67</td>
<td>123</td>
<td>436</td>
<td>68</td>
<td>112</td>
<td>424</td>
<td>185</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>377</td>
<td>409</td>
<td>291</td>
<td>101</td>
<td>452</td>
<td>412</td>
<td>281</td>
<td>473</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>17</td>
<td>50</td>
<td>16</td>
<td>14</td>
<td>16</td>
<td>25</td>
<td>31</td>
<td>35</td>
<td>48</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>41</td>
<td>18</td>
<td>41</td>
<td>42</td>
<td>29</td>
<td>0</td>
<td>32</td>
<td>45</td>
<td>6</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>33</td>
<td>5</td>
<td>47</td>
<td>29</td>
<td>12</td>
<td>3</td>
<td>47</td>
<td>600</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### data_20.data

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>8880</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>427</td>
<td>76</td>
</tr>
<tr>
<td>27</td>
<td>412</td>
<td>116</td>
</tr>
<tr>
<td>377</td>
<td>409</td>
<td>101</td>
</tr>
<tr>
<td>44</td>
<td>17</td>
<td>50</td>
</tr>
<tr>
<td>31</td>
<td>41</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>33</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>486</th>
<th>211</th>
<th>162</th>
<th>459</th>
<th>280</th>
<th>236</th>
<th>195</th>
<th>486</th>
<th>316</th>
</tr>
</thead>
<tbody>
<tr>
<td>271</td>
<td>412</td>
<td>316</td>
<td>456</td>
<td>67</td>
<td>123</td>
<td>436</td>
<td>68</td>
<td>112</td>
<td>424</td>
</tr>
<tr>
<td>377</td>
<td>409</td>
<td>291</td>
<td>101</td>
<td>452</td>
<td>412</td>
<td>281</td>
<td>473</td>
<td>1</td>
<td>316</td>
</tr>
<tr>
<td>44</td>
<td>17</td>
<td>50</td>
<td>16</td>
<td>14</td>
<td>16</td>
<td>25</td>
<td>31</td>
<td>35</td>
<td>48</td>
</tr>
<tr>
<td>31</td>
<td>41</td>
<td>18</td>
<td>41</td>
<td>42</td>
<td>29</td>
<td>0</td>
<td>32</td>
<td>45</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>33</td>
<td>5</td>
<td>47</td>
<td>29</td>
<td>12</td>
<td>3</td>
<td>47</td>
<td>600</td>
<td></td>
</tr>
</tbody>
</table>

### data_21.data

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>15000</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>143</td>
<td>204</td>
</tr>
<tr>
<td>81</td>
<td>104</td>
<td>374</td>
</tr>
<tr>
<td>214</td>
<td>480</td>
<td>205</td>
</tr>
<tr>
<td>74</td>
<td>419</td>
<td>182</td>
</tr>
<tr>
<td>134</td>
<td>356</td>
<td>173</td>
</tr>
<tr>
<td>494</td>
<td>466</td>
<td>346</td>
</tr>
<tr>
<td>36</td>
<td>38</td>
<td>32</td>
</tr>
<tr>
<td>21</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>30</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>34</td>
<td>47</td>
<td>19</td>
</tr>
<tr>
<td>50</td>
<td>25</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>96</th>
<th>151</th>
<th>56</th>
<th>493</th>
<th>345</th>
<th>450</th>
<th>10</th>
<th>21</th>
<th>445</th>
</tr>
</thead>
<tbody>
<tr>
<td>81</td>
<td>104</td>
<td>154</td>
<td>394</td>
<td>239</td>
<td>294</td>
<td>131</td>
<td>477</td>
<td>387</td>
<td>279</td>
</tr>
<tr>
<td>214</td>
<td>480</td>
<td>452</td>
<td>11</td>
<td>414</td>
<td>161</td>
<td>465</td>
<td>369</td>
<td>116</td>
<td>155</td>
</tr>
<tr>
<td>74</td>
<td>419</td>
<td>263</td>
<td>10</td>
<td>385</td>
<td>168</td>
<td>409</td>
<td>27</td>
<td>193</td>
<td>295</td>
</tr>
<tr>
<td>134</td>
<td>356</td>
<td>281</td>
<td>358</td>
<td>245</td>
<td>112</td>
<td>161</td>
<td>360</td>
<td>3</td>
<td>217</td>
</tr>
<tr>
<td>494</td>
<td>466</td>
<td>397</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>38</td>
<td>23</td>
<td>35</td>
<td>40</td>
<td>44</td>
<td>40</td>
<td>35</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>43</td>
<td>35</td>
<td>37</td>
<td>32</td>
<td>14</td>
<td>15</td>
<td>7</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>30</td>
<td>27</td>
<td>42</td>
<td>14</td>
<td>11</td>
<td>29</td>
<td>16</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>39</td>
<td>30</td>
<td>5</td>
<td>27</td>
<td>8</td>
<td>6</td>
<td>21</td>
<td>49</td>
</tr>
<tr>
<td>34</td>
<td>47</td>
<td>41</td>
<td>4</td>
<td>22</td>
<td>19</td>
<td>20</td>
<td>42</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>25</td>
<td>8</td>
<td>0</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1000
Appendix B: Copy of the java code created
Class sort

/**
 * Sort the data based on various rules (sort largest value, sort by largest contraints, etc)
 */

import java.util.*;
import java.io.*;
import java.util.Arrays;
import java.util.Random.*/;
import java.math.BigInteger; //used for the combinations
import java.lang.Math;

public class sort {
    /**Constructor**/
    public sort()
    {
    }

    public void sortData()
    {
    }

    /**Methods**/

    /**Print out final information**/

    //Print out information about each data set
    //This printing method is called by each picking method
    public static void printInfo(int num_items_packed, int sum_items_packed, int opt_soln, int [] packed_items_array) {
        System.out.println("No more items will fit in the remaining space. We packed " + num_items_packed + " items."
            + "\n" + "The sum value of the items packed is: " + sum_items_packed
            + "\n" + "The Optimal Solution is " + opt_soln + ", which is a "
            + "\n" + (float)(opt_soln-sum_items_packed)/(opt_soln)*100 + " percent difference."
        );

        System.out.print("We packed the following items: ");
        for (int j = 0; j<packed_items_array.length; j++){
            if (packed_items_array[j] != 0){
                System.out.print(packed_items_array[j] + " ");
            }
        }
    }
public static void sortSmall(int no_variables, int no_constraints, int[][] test_prob_array, int[] test_prob_array_objectVals, int opt_soln, int[] test_prob_array_constraints)
{
    int[] temp_array_constraints = new int[no_variables];
    for (int i = 0; i<no_variables; i++)
    {
        temp_array_constraints[i] = test_prob_array[0][i];
    }

    //Add object Vals array to last row of 2d test_prob_array
    for (int i = 0; i<no_variables; i++)
    {
        test_prob_array[no_constraints][i] = test_prob_array_objectVals[i];
    }

    System.out.println("The maximum of constraint 1 is " + test_prob_array_constraints[0]);

    //sort array of constraints using built in java class "sort"
    Arrays.sort(temp_array_constraints, 0, no_variables);

    //Move the object values to their corresponding, sorted constraint
    int[][] temp_array_test_prob_array = new int[no_constraints+1][no_variables];
    for (int index1 = 0; index1<no_variables; index1++)
    {
        for (int index = 0; index <no_variables; index++)
        {
            if (temp_array_constraints[index] == test_prob_array[0][index1])
            {
                for (int i = 0; i<no_constraints+1; i++)
                {
                    temp_array_test_prob_array[i][index1] = test_prob_array[i][index];
                }
            }
        }
    }

    //Assign the temporary array back to the test_problem array

test_prob_array = temp_array_test_prob_array;

System.out.println("Pick the objects with smallest constraints first");

//Print out the information for this data set and picking method
pickItems.printItemsPicked(test_prob_array_constraints, test_prob_array, no_variables, no_constraints, opt_soln);
};//end sortSmall

/***Pick smallest values first****/

public static void sortValsSmall(int no_variables, int no_constraints, int [][] test_prob_array,
int [] test_prob_array_objectVals, int opt_soln, int [] test_prob_array_constraints)
{

//Add object Vals array to last row of 2d test_prob_array
for (int i = 0; i<no_variables; i++){
    test_prob_array[no_constraints][i] = test_prob_array_objectVals[i];
}

//Pick Items based on object values
System.out.println("The maximum of constraint 1 is "+test_prob_array_constraints[0]);

//create a temporary array of constraints and fill it with the first constraint
int[] temp_array_constraints = new int [no_variables];
for (int i = 0; i<no_variables; i++){
    temp_array_constraints[i] = test_prob_array[1][i];
}

//sort array of constraints using built in java class "sort"
//sort based on last line, which is the object values
Arrays.sort(temp_array_constraints,0,no_variables);

int[][] temp_array_test_prob_array = new int [no_constraints+1][no_variables];

for (int index1 = 0; index1<no_variables; index1++){
    for (int index = 0; index <no_variables; index++){
        if (temp_array_constraints[index] == test_prob_array[1][index1]){
            for (int i = 0; i<no_constraints+1; i++){
                temp_array_test_prob_array[i][index1] = test_prob_array[i][index1];
            }
        }
    }
}
test_prob_array = temp_array_test_prob_array;

System.out.println("Pick the Objects with the smallest value first");

//Print out the information for this data set and picking method
pickItems.printItemsPicked(test_prob_array_constraints, test_prob_array, no_variables, no_constraints, opt_soln);
} //end sortValsSmall

/***Pick largest constraints firstt****/
public static void sortLarge(int no_variables, int no_constraints, int [][]test_prob_array,
   int [] test_prob_array_objectVals, int opt_soln, int [] test_prob_array_constraints)
{
   int[] temp_array_constraints = new int [no_variables];

   for (int i = 0; i<no_variables; i++){
      temp_array_constraints[i] = test_prob_array[0][i];
   } //end for loop

   //Add object Vals array to last row of 2d test_prob_array
   for (int i =0; i<no_variables; i++)
   {
      test_prob_array[no_constraints][i] = test_prob_array_objectVals[i];
   } //end for

   System.out.println("The maximum of constraint 1 is " + test_prob_array_constraints[0]);

   //sort array of constraints using built in java class "sort"
   Arrays.sort(temp_array_constraints,0,no_variables);

   //Move the object values to their corresponding, sorted constraint s
   int[][] temp_array_test_prob_array = new int [no_constraints+1][no_variables];

   for (int index1 = 0;index1<no_variables; index1++)
   {
      for (int index = 0; index <no_variables; index++)
      {
         if (temp_array_constraints[index] == test_prob_array[0][index])
            temp_array_test_prob_array[index1][index] = test_prob_array_objectVals[index);
      } //end for
   } //end for
   }
```
for (int i = 0; i < no_constrants + 1; i++) {
    temp_array_test_prob_array[i][index] = test_prob_array[i][index1];
} //end for
} //end if
} //end for
} //end for

//Assign the temporary array back to the test_problem array
test_prob_array = temp_array_test_prob_array;

//reverse the array so that the largest constraints are first
int[][] temp_array_test_prob_array2 = new int [no_constraints + 1][no_variables];
for (int i = 0; i < no_constraints + 1; i++) {
    for (int j = 0; j < no_variables; j++) {
        temp_array_test_prob_array2[i][j] = test_prob_array[i][no_variables - 1 - j];
    } //end second for
} //end first for

test_prob_array = temp_array_test_prob_array2;

System.out.println("Pick the Objects with the largest constraints first");

//Print out the information for this data set and picking method
pickItems.printItemsPicked(test_prob_array_constraints, test_prob_array, no_variables, no_constrants, opt_soln);
} //end sortValsSmall

/***Pick Largest values first***/
public static void sortValsLarge(int no_variables, int no_constrants, int [][] test_prob_array,
    int [] test_prob_array_objectVals, int opt_soln, int [] test_prob_array_constraints) {

    //Add object Vals array to last row of 2d test_prob_array
    for (int i = 0; i < no_variables; i++) {
        test_prob_array[no_constraints][i] = test_prob_array_objectVals[i];
    } //end for

    //Pick Items based on object values
    System.out.println("The maximum of constraint 1 is " + test_prob_array_constraints[0]);

} //end sortValsLarge
//create a temporary array of constraints and fill it with the
//first constraint
int[] temp_array_constraints = new int[no_variables];
for (int i = 0; i < no_variables; i++) {
    temp_array_constraints[i] = test_prob_array[1][i];
}

//sort array of constraints using built in java class "sort"
//sort based on last line, which is the object values
Arrays.sort(temp_array_constraints, 0, no_variables);

int[][] temp_array_test_prob_array = new int[no_constraints+1][no_variables];

for (int index1 = 0; index1 < no_variables; index1++) {
    for (int index = 0; index < no_variables; index++) {
        if (temp_array_constraints[index] == test_prob_array[1][index1]) {
            for (int i = 0; i < no_constraints+1; i++) {
                temp_array_test_prob_array[i][index] = test_prob_array[i][index1];
            }
        }
    }
}

for (int i = 0; i < no_constraints+1; i++) {
    for (int j = 0; j < no_variables; j++) {
        temp_array_test_prob_array[i][j] = test_prob_array[i][no_variables-1-j];
    }
}

test_prob_array = temp_array_test_prob_array;

int[][] temp_array_test_prob_array1 = new int[no_constraints+1][no_variables];

for (int i = 0; i < no_constraints+1; i++) {
    for (int j = 0; j < no_variables; j++) {
        temp_array_test_prob_array1[i][j] = test_prob_array[i][no_variables-1-j];
    }
}

test_prob_array = temp_array_test_prob_array1;

System.out.println("Pick the largest Objects first");
pickItems.printItemsPicked(test_prob_array_constraints, test_prob_array, no_variables, no_constraints, opt_soln);

//end sortValsLarge
public static void sortValsRand(int no_variables, int no_constraints, int[] test_prob_array, int[] test_prob_array_objectVals, int opt_soln, int[] test_prob_array_constraints) {
    // Add object Vals array to last row of 2d test_prob_array
    for (int i = 0; i < no_variables; i++) {
        test_prob_array[no_constraints][i] = test_prob_array_objectVals[i];
    }
    // end for
    int[] temp_packed_items_array = new int[no_variables];

    /** This method fills up an array of no_variables with random indices**/
    // size of the array to be filled with random indices
    int size = no_variables;

    // array that stores indices
    int[] randomIndices = new int[size];

    // for fills the array indices with intial values 0 - array size
    for (int i = 0; i < size; i++)
        randomIndices[i] = i;

    Random random = new Random();

    // rearranges values in array so that they are randomized and indices are not
    // repeated
    for (int i = 0; i < size; i++)
    {
        // boolean used to track if an indice is not a repeated number
        boolean unique = false;
        int randomNo = 0;
        // while loop guarantees a unique indice
        while (!unique)
        {
            unique = true;
            randomNo = random.nextInt(size);
            // checks previously assigned values
            for (int j = 0; j < i; j++)
            {
                if (randomIndices[j] == randomNo)
                {
                    unique = false;
                }
            }
        }
    }
}
public static void lookAheadSortSmallVal(int no_variables, int no_constraints, int [][]test_prob_array, int [][]test_prob_array_objectVals, int opt_soln, int [] test_prob_array_constraints) {
    //Add object Vals array to last row of 2d test_prob_array
    for (int i = 0; i < no_variables; i++)
        test_prob_array[no_constraints][i] = test_prob_array_objectVals[i];

    //Pick Items based on object values
    System.out.println("The maximum of constraint 1 is " + test_prob_array_constraints[0]);

    //create a temporary array of constraints and fill it with the first constraint
    int[] temp_array_constraints = new int [no_variables];
    for (int i = 0; i < no_variables; i++)
        temp_array_constraints[i] = test_prob_array[1][i];

    //sort array of constraints using built in java class "sort"
    //sort based on last line, which is the object values
    Arrays.sort(temp_array_constraints, 0, no_variables);

    int[][] temp_array_test_prob_array = new int [no_constraints+1][no_variables];
    for (int index1 = 0; index1 < no_variables; index1++)
        for (int index = 0; index < no_variables; index++)
if (temp_array_constraints[index] == test_prob_array[1][index]){
    for (int i = 0; i<no_constraints+1; i++){
        temp_array_test_prob_array[i][index] = test_prob_array[i][index];
    } //end third for
} //end if
} //end second for
} //end first for

test_prob_array = temp_array_test_prob_array;

System.out.println("Pick the objects with smallest values first");

System.out.println("Use backtracking method to unpack items");
//Print out the information for this data set and picking method
System.out.println("-----------------------------");

int remaining_space = test_prob_array_constraints[0];
int row = 0;
int num_items_packed = 0;
int sum_items_packed = 0;
int [] packed_items_array = new int [no_variables];

for(int i =0; i<no_variables;i++){
    //if there's more space, pick another item
    if (remaining_space-(test_prob_array[row][i]) >0){
        System.out.println("The item picked is: "+test_prob_array[no_constraints][i]);
        packed_items_array[i] = test_prob_array[no_constraints][i] ;
        System.out.println("The remaining space is: "+(remaining_ space-test_prob_array[row][i])+"\n");
        //calculate the remaining space, increment number of items packed, find sum (value) of items packed
        remaining_space = remaining_space - (test_prob_array[row][i]);
    } //end if
    else if(i!=0) {
        remaining_space = remaining_space + test_prob_array[row][i-1];
        packed_items_array[i-1] = 0;
        System.out.println("The item removed is: "+ test_prob_array[y[no_constraints][i-1]]+"\n");
    i--;
}
num_items_packed = packed_items_array.length + 1;
for (int i=0; i<num_items_packed-1; i++)
    sum_items_packed = sum_items_packed + packed_items_array[i];

sort.printlnfo(num_items_packed, sum_items_packed, opt_soln, packed_items_array);

/***Sort objects by value, and use a lookahead method to pick items***/

public static void lookAheadSortLargeVal(int no_variables, int no_constraints, int [][] test_prob_array,
int [] test_prob_array_objectVals, int opt_soln, int [][] test_prob_array_constraints)
{
    //Add object Vals array to last row of 2d test_prob_array
    for (int i=0; i<no_variables; i++)
        test_prob_array[no_constraints][i] = test_prob_array_objectVals[i];

    //Pick Items based on object values
    System.out.println("The maximum of constraint 1 is "+ test_prob_array_constraints[0]);

    //create a temporary array of constraints and fill it with the first constraint
    int[] temp_array_constraints = new int [no_variables];
    for (int i = 0; i<no_variables; i++)
        temp_array_constraints[i] = test_prob_array[1][i];

    //sort array of constraints using built in java class "sort"
    //sort based on last line, which is the object values
    Arrays.sort(temp_array_constraints, 0, no_variables);

    int[][] temp_array_test_prob_array = new int [no_constraints+1][no_variables];
    for (int index1 = 0; index1<no_variables; index1++)
        for (int index = 0; index <no_variables; index++)
            if (temp_array_constraints[index] == test_prob_array[1][index1])
                temp_array_test_prob_array[index1][index] = test_prob_array[index1][index1];
```java
for (int i = 0; i < no_constraints + 1; i++) {
    for (int j = 0; j < no_variables; j++) {
        temp_array_test_prob_array1[i][j] = test_prob_array[i][no_variables - 1 - j];
    }
}
```

```
System.out.println("Pick the objects with Largest values first");
System.out.println("Use backtracking method to unpack items");
//Print out the information for this data set and picking method
System.out.println("-----------------------");
```

```
int remaining_space = test_prob_array_constraints[0];
int row = 0;
int num_items_packed = 0;
int sum_items_packed = 0;
int[] packed_items_array = new int[no_variables];
```

```
for (int i = 0; i < no_variables; i++) {
    //if there's more space, pick another item
    if (remaining_space - (test_prob_array[row][i]) > 0) {
        System.out.println("The item picked is: " + test_prob_array[no_constraints][i] + ":
          packed_items_array[i] = test_prob_array[no_constraints][i] + ");
        System.out.println("The remaining space is: " + (remaining_space - test_prob_array[row][i]) + ");
```

```
//calculate the remaining space, increment number of items packed
remaining_space = remaining_space - (test_prob_array[row][i]);
```
```java
class sort (continued)

    } //end if
    else if( i!=0 ) {
        remaining_space = remaining_space + test_prob_array[row][i-1];
        packed_items_array[i-1] = 0;
        System.out.println("The item removed is: " + test_prob_array[no_constraints][i-1]+
                        "\n");
        i--;
    }
} //end for loop

num_items_packed = packed_items_array.length + 1;
for ( int i=0; i<num_items_packed-1; i++ )
    sum_items_packed = sum_items_packed + packed_items_array[i];

sort.printInfo(num_items_packed, sum_items_packed, opt_soln, packed_items_array);
} //end lookAheadSortLargeVal

/***calculate all combinations***/
public static void allCombos(int no_variables, int no_constraints,
    int [][]test_prob_array,
    int [] test_prob_array_objectVals, int opt_soln, int [] test_prob_array_constraints)

    //Add object Vals array to last row of 2d test_prob_array
    for ( int i =0; i<no_variables; i++)
        test_prob_array[no_constraints][i] = test_prob_array_objectVals[i];
} //end for

int remaining_space = test_prob_array_constraints[0];
int row = 0;
//int A = 0; //used for filling in allCombos array
int [] bestCombo_array = new int [no_variables]; //array to store the best combination
int [] temp_bestCombo_array = new int [no_variables]; //temp array to store combinations

int sum_vals = 0;
int sum_cons = 0;
int best_sum_vals = 0;
int best_sum_cons = 0;
int array_length = 0;
int [] indices;
```
for (int j=0; j<no_variables+1; j++) {
    CombinationGenerator x = new CombinationGenerator (test_prob_array.objectVals.length, j);

    StringBuffer combination;

    while (x.hasMore ()) {
        combination = new StringBuffer ();
        indices = x.getNext ();
        int B = 0;
        sum_vals=0;
        sum_cons=0;
        for (int i = 0; i < indices.length; i++) {
            temp_bestCombo_array [B] = test_prob_array.objectVals[indices[i]];
            B++;
            sum_vals += test_prob_array.objectVals[indices[i]];
            sum_cons += test_prob_array[0][indices[i]];
        }
    } //end second for loop

    System.out.println("bestCombo_array is ");
    for (int k = 0; k<bestCombo_array.length; k++) {
        System.out.print(bestCombo_array[k] + " " );
    } //end for loop
    System.out.println(best_sum_vals);
}

} //end first for loop

System.out.println("\n\n"+"The Best Combination has an object value sum of " + best_sum_vals + "\n" +
    "which has a constraint sum of " + best_sum_cons + "\n" +
    "We packed the following items: ");

for (int j = 0; j<bestCombo_array.length; j++) {
    if (bestCombo_array[j] != 0)
        System.out.print(bestCombo_array[j] + " ");
} //end for loop

System.out.println();

} //end allCombos
Class pickItems

/**
 * Pick Items is a class which inputs the picking method
 * 1. Greedy Algorithm: pick the object with the smallest constraint value first
 * 2. Greedy Algorithm: pick the object with the smallest value first
 * 3. Greedy Algorithm: pick the object with the largest constraint value first
 * 4. Greedy Algorithm: pick the object with the largest value first
 * 5. Random Picker: pick items randomly using a random number generator
 * 6. Look ahead method: compare the current object to be picked with the next object
 *    (sorted by smallest constraint)
 * 7. Look ahead method: compare the current object to be picked with the next object
 *    (sorted by smallest object value)
 * 8. Look ahead method: compare the current object to be picked with the next object
 *    (sorted by largest constraint)
 * 9. Look ahead method: compare the current object to be picked with the next object
 *    (sorted by largest object value)
 * 10. Compute all possible combinations of objects and output 5 best and 5 worst combinations
 */

import java.util.*;
import java.io.*;
import java.util.Arrays;
import java.util.Random.*;

public class pickItems
{
    //private variables
    int selection2 = 0;

    public pickItems()
    {
    }

    /****Methods****/
    public static void pick (int [] test_prob_array_constraints, int no_variables, int [][] test_prob_array,
                           int no_constraints, int opt_soln, int [] test_prob_array_objectVals){
System.out.println("Enter a number:");
System.out.println("1: Greedy Algorithm: Pick smallest constraint first" + "\n" +
"2: Greedy Algorithm: Pick smallest value first" + "\n" +
"3: Greedy Algorithm: Pick largest constraint first" + "\n" +
"4: Greedy Algorithm: Pick largest value first" + "\n" +
"5: Random Picker: Pick items randomly, using the random number generator" + "\n" +
"7: Look ahead method: compare the current object to be picked with the next object (sorted by smallest object value)" + "\n" +
"8: Look ahead method: compare the current object to be picked with the next object (sorted by largest object value)" + "\n" +
"11: Compute all possible combinations of objects and output best combinations" + "\n"
);

Scanner scanner1 = new Scanner(System.in);
int selection2 = scanner1.nextInt();

switch (selection2)
{
    case 1:
        sort.sortSmall(no_variables, no_constraints, test_prob_array, 
test_prob_array_objectVals, opt_soln, test_prob_array_constraints);
        break;
    case 2:
        sort.sortValsSmall(no_variables, no_constraints, 
        test_prob_array, test_prob_array_objectVals, opt_soln, test_prob_array_constraints);
        break;
    case 3:
        sort.sortLarge(no_variables, no_constraints, test_prob_array, 
        test_prob_array_objectVals, opt_soln, test_prob_array_constraints);
        break;
    case 4:
### Class `pickItems` (continued)

```
    public static void printItemsPicked (int [] test_prob_array_constraints, int [][] test_prob_array, int no_variables,
                                   int no_constraints, int opt_soln) {
        System.out.println("--------------------------");

        int remaining_space = test_prob_array_constraints[0];
        int row = 0;
        int num_items_packed = 0;
        int sum_items_packed = 0;
        int [] packed_items_array = new int [no_variables];

        sort.sortValsLarge(no_variables, no_constraints, test_prob_array,
                           test_prob_array_objectVals, opt_soln, test_prob_array_constraints);
        break;

        case 5:
            sort.sortValsRand(no_variables, no_constraints, test_prob_array,
                              test_prob_array_objectVals, opt_soln, test_prob_array_constraints);
            break;

        case 7:
            sort.lookAheadSortSmallVal(no_variables, no_constraints, test_prob_array,
                                        test_prob_array_objectVals, opt_soln, test_prob_array_constraints);
            break;

        case 8:
            sort.lookAheadSortLargeVal(no_variables, no_constraints, test_prob_array,
                                        test_prob_array_objectVals, opt_soln, test_prob_array_constraints);
            break;

        case 11:
            sort.allCombos(no_variables, no_constraints, test_prob_array,
                           test_prob_array_objectVals, opt_soln, test_prob_array_constraints);
            break;

    } //end switch
} //end pickItems
```
for(int i = 0; i< no_variables; i++){

    //if there's more space, pick another item
    if (remaining_space-(test_prob_array[row][i]) > 0) {
        System.out.println("The item picked is: "+ test_prob_array[no_constraints][i]);
        packed_items_array[i] = test_prob_array[no_constraints][i];
        System.out.println("The remaining space is: "+ (remaining_space-test_prob_array[row][i]) +"\n");

        //calculate the remaining space, increment number of items packed, find sum (value) of items packed
        remaining_space = remaining_space - (test_prob_array[row][i]);
        if (packed_items_array[i] != 0)
            num_items_packed++;
        sum_items_packed += test_prob_array[no_constraints][i];
    }
    //end if
    // else {
    //    break;
    //}
} //end for loop

sort.printInfo(num_items_packed, sum_items_packed, opt_solution, packed_items_array);
}//end printItemsPicked

}//end class pickItems
import java.util.*;
import java.io.*;
import java.util.Arrays;
import java.util.Random.*;

public class Driver {
    public Driver() {
    }

    /** Methods **/
    public static void getData() throws FileNotFoundException, IOException {
        String pickAgain = "Y";
        while (pickAgain.compareTo("y") == 0 || pickAgain.compareTo("Y") == 0 ||
                pickAgain.compareTo("yes") == 0 || pickAgain.compareTo("YES") == 0) {
            System.out.println("Enter the number of the test problem:");
            System.out.println("1: Multiple values, based on mknap file with first constraint" + "\n" +
                "2: Data set we created the first day" + "\n" +
                "3: Data set where values are randomly chosen, no correspondence" + "\n" +
                "4: Optimal solution found by taking smallest items first" + "\n" +
                "5: Multiple values, based on mknap file problem 2 with first constraint" + "\n" +
                "6: Multiple values, based on mknap file problem 3 with first constraint" + "\n" +
                "7: Multiple values, based on mknap file problem 4 with first constraint" + "\n" +
                "8: Optimal solution found by taking largest items first" + "\n" +
                "9: Large Data set 1" + "\n" +
                "10: Large Data set 2" + "\n" +
                "11 Clumps of items: 4 clumps of similar-valued objects and clumped constraints randomly generated" + "\n" +
                "12: Data set where values are randomly chosen, no correspondence" + "\n" +
                "13: Obj vals randomly generated, constraints randomly generated" + "\n" +
                "14: Optimal solution found by taking smallest items first, or smallest weights first" + "\n" +
    }
    }
}
15: Optimal solution found by taking largest items first, or smallest weights first
"16: Obj vals randomly generated; clumps of weights (in 5's and 10's)"
"17: 10 randomly generated items"
"18: 20 randomly generated items"
"19: 30 randomly generated items"
"20: 40 randomly generated items"
"21: 50 randomly generated items"

Scanner scanner1 = new Scanner(System.in);
int selection = scanner1.nextInt();

switch (selection) {
    case 1:
        getInput("data_1.data");
        break;
    case 2:
        getInput("data_2.data");
        break;
    case 3:
        getInput("data_3.data");
        break;
    case 4:
        getInput("data_4.data");
        break;
    case 5:
        getInput("data_5.data");
        break;
    case 6:
        getInput("data_6.data");
        break;
    case 7:
        getInput("data_7.data");
        break;
    case 8:
        getInput("data_8.data");
        break;
    case 9:
```
getInput("data_9.data");
break;

case 10:
    getInput("data_10.data");
    break;

case 11:
    getInput("data_11.data");
    break;

case 12:
    getInput("data_12.data");
    break;

case 13:
    getInput("data_13.data");
    break;

case 14:
    getInput("data_14.data");
    break;

case 15:
    getInput("data_15.data");
    break;

case 16:
    getInput("data_16.data");
    break;

case 17:
    getInput("data_17.data");
    break;

case 18:
    getInput("data_18.data");
    break;

case 19:
    getInput("data_19.data");
    break;

case 20:
    getInput("data_20.data");
    break;

case 21:
    getInput("data_21.data");
```
break;
  } //end switch

  //Pick Again
  System.out.println("Would you like to select another data set and picking method? Y/N " + "\n");
  Scanner scanner2 = new Scanner(System.in);
  pickAgain = scanner2.next();
  //pickAgain = "y";
  }//end while

  System.exit(0);
  }//end getData

  public static void getlnput (String file) throws FileNotFoundException, IOException {

    /**Public Variables**/
    int no_test_probs;  //Data set number
    int no_variables;  //Number of variables in data set
    int no_constraints;  //Number of constraints in data set
    int opt_soln;  //Value of the optimal solution

    //Test problem array
    int [] test_prob_array_constraints;
    int [] test_prob_array_objectVals;
    int [][] test_prob_array;

    //open the Scanner
    Scanner scanner = new Scanner(new File(file));

    //get details from data set
    no_test_probs = scanner.nextInt();
    no_variables = scanner.nextInt();
    no_constraints = scanner.nextInt();
    opt_soln = scanner.nextInt();

    System.out.println("Number of test problems " + no_testProbe + "\n" + "Number of variables " + no_variables + "\n" + "Number of constraints " + no_constraints + "\n" + "Optimal Solution " + opt_soln + "\n"
//Get object values
test_prob_array_objectVals = new int [no_variables];
for (int k=0; k<no_variables; k++) {
    test_prob_array_objectVals[k] = scanner.nextInt();
}//end for loop

//Get test problem array
for (int i=0; i<no_constraints+1; i++) {
    for (int j=0; j<no_variables; j++) {
        test_prob_array[i][j] = scanner.nextInt();
    //end second for loop
    }//end first for loop

//Get contraint array
for (int k=0; k<no_constraints; k++) {
    test_prob_array_constraints[k] = scanner.nextInt();
}//end for loop

//input done, so close the scanner
scanner.close();

    pickItems.pick(test_prob_array_constraints, no_variables, test_prob_array, no_constraints, opt_soln, test_prob_array_objectVals);

    }//end getlnput
}//end main
import java.math.BigInteger;

public class CombinationGenerator {

    private int[] a;
    private int n;
    private int r;
    private BigInteger numLeft;
    private BigInteger total;

    // Constructor
    // ---------------
    public CombinationGenerator (int n, int r) {
        if (r > n) {
            throw new IllegalArgumentException ();
        }
        if (n < 1) {
            throw new IllegalArgumentException ();
        }
        this.n = n;
        this.r = r;
        a = new int[r];
        BigInteger nFact = getFactorial (n);
        BigInteger rFact = getFactorial (r);
        BigInteger nminusrFact = getFactorial (n - r);
        total = nFact.divide (rFact.multiply (nminusrFact));
        reset ();
    }

    // Reset
    // ------
    public void reset () {
        for (int i = 0; i < a.length; i++) {
            a[i] = i;
        }
        numLeft = new BigInteger (total.toString ());
    }

    // Return number of combinations not yet generated
    // --------------------------------------------------
public BigInteger getNumLeft () {  
    return numLeft;
}

//-----------------------------  
// Are there more combinations?  
//-----------------------------  

public boolean hasMore () {  
    return numLeft.compareTo (BigInteger.ZERO) == 1;
}

//-------------------------------  
// Return total number of combinations  
//-------------------------------  

public BigInteger getTotal () {  
    return total;
}

//--------------------------  
// Compute factorial  
//--------------------------  

private static BigInteger getFactorial (int n) {  
    BigInteger fact = BigInteger.ONE;  
    for (int i = n; i > 1; i--) {  
        fact = fact.multiply (new BigInteger (Integer.toString (i)));
    }  
    return fact;
}

//-------------------------------  
// Generate next combination (algorithm from Rosen p. 286)  
//-------------------------------  

public int[] getNext () {  
    if (numLeft.equals (total)) {  
        numLeft = numLeft.subtract (BigInteger.ONE);  
        return a;
    }
    int i = r - 1;  
    while (a[i] == n - r + i) {  
        i--;
    }  
    a[i] = a[i] + 1;
for (int j = i + 1; j < r; j++) {
    a[j] = a[i] + j - i;
}

numLeft = numLeft.subtract (BigInteger.ONE);
return a;
}