Lab 1

This lab is an exploration of the Knapsack Problem, mentioned in the slides for Lesson 1. In the slides, the *Knapsack decision problem* is given; this is a version of the problem that returns "true" or "false." The *Knapsack optimization problem* is the following:

Knapsack Optimization Problem Given a set $S = \{s_0, s_1, \ldots, s_{n-1}\}$ of n items with positive integer weights given by $w[] = \{w_0, w_1, \ldots, w_{n-1}\}$ and positive integer values $v[] = \{v_0, v_1, \ldots, v_{n-1}\}$ and a maximum weight W (a positive integer), find $T \subseteq \{s_0, s_1, \ldots, s_{n-1}\}$ so that $\sum_{s_t \in T} v_t$ is maximal (the *maximum benefit*), subject to the constraint that $\sum_{s_t \in T} w_t \leq W$. Such a set T is called a *solution* to the knapsack problem; note that a solution is not just a set T over which the sum of the weights is $\leq W$; T must provide the highest sum of the $v[i]$ as well. Sometimes, one thinks of a *solution* as being the greatest possible sum of values from $v[]$ for which the constraint is satisfied.

Problem 1. Formulate your own procedure for solving this problem. Think of it as a Java method `knapsack` that accepts as input $S, v[], w[], W$ and outputs a subset S_0 of S with the property that the sum of the v_i for s_i in S_0 is maximal for which the sum of w_i for $s_i \in S_0$ is still $\leq W$. If you have time, implement your idea as a Java method.

Problem 2. *Greedy Strategies.* See if you can solve knapsack problems using one of the following *greedy* strategies. With a greed strategy, at each step in an algorithm a value that is optimal at that time is chosen. To illustrate the methods, consider the following example: $S = \{0, 1, 2\}, v[] = \{1, 3, 4\}$, $w[] = \{1, 2, 4\}$, $W = 4$. We want to find a subset of S whose value is maximum but whose total weight does not exceed W.

(1) Try arranging S in increasing order of weight. Then load the knapsack with items from S until it becomes impossible to add any more because of the weight restriction. In the example, if S is arranged in this way, $S, v[]$, and $w[]$ remain the same: $S = \{0, 1, 2\}$, $v[] = \{1, 3, 4\}, w[] = \{1, 2, 4\}$. When we add s_0 to the knapsack, we have a total weight of 1; then if we add s_1, we have a total weight of $1 + 2 = 3$. We cannot add s_2 because of the weight limit. So our algorithm in this case outputs the subset $\{s_0, s_1\}$, or simply $\{0, 1\}$. Check that this result is indeed correct. Then answer this question: Can you invent other values for $S, v[], w[], W$ for which this strategy will not give a correct answer? Can you see this approach is a *greedy strategy*, attempting to fill the knapsack with items of smallest possible weight at each step?

(2) Try arranging S in decreasing order of value. Then load the knapsack with items from S until it becomes impossible to add any more because of the weight restriction. In the

example, if S is arranged by value, the original ordering of S is reversed—we have $S = \{2, 1, 0\}, v[] = \{4, 3, 1\}$, $w[] = \{4, 2, 1\}$, $W = 4$. s_0 in this case has weight 4, so we can pick no more items. The output in this case would be $\{s_0\}$ or just $\{0\}$. Once again, the best possible value was achieved. Do you see that this is another kind of greedy strategy, in this case going for the best possible value at each step. Again, answer this question: Can you invent other values for $S, v[], w[], W$ for which this strategy will not give a correct answer?

(3) Try arranging S in decreasing order of value per weight. For each i, let $b_i = v_i/w_i$. Then we compute: $b_0 = 1, b_1 = 1.5, b_2 = 1$. If we arrange by decreasing order of the b_i, we get $S = \{1, 0, 2\}, v[] = \{3, 1, 4\}, w[] = \{2, 1, 4\}$. Now load the knapsack with items from S until it becomes impossible to add any more because of the weight restriction. s_0 has weight 2, so we add it to the knapsack. Here, $s_1 = 0$ has weight 1, so we can also add that to the knapsack. We cannot add the final item because of the weight restriction. The values of s_0 and s_1 total $3 + 1 = 4$. We have once again obtained a correct result. Can you invent other values for $S, v[], w[], W$ for which this strategy will not give a correct answer?

Problem 3. Try answering the following:

(1) You are trying to solve a knapsack problem; as always, you are given $S = \{s_0, s_1, \ldots, s_{n-1}\}$, $w[] = \{w_0, w_1, \ldots, w_{n-1}\}$, $v[] = \{v_0, v_1, \ldots, v_{n-1}\}$ and a maximum weight W. Is it possible that an optimal solution will not make use of item s_{n-1} (in other words, that a particular optimal solution S_0 does not contain the item s_{n-1})? Give an example to illustrate your idea.

(2) You are trying to solve a knapsack problem; as always, you are given $S = \{s_0, s_1, \ldots, s_{n-1}\}$, $w[] = \{w_0, w_1, \ldots, w_{n-1}\}$, $v[] = \{v_0, v_1, \ldots, v_{n-1}\}$ and a maximum weight W. Suppose that an optimal solution $S_0 \subseteq S$ makes use of the item s_{n-1} (in other words, $s_{n-1} \in S_0$). Suppose now that we remove s_{n-1} from S, giving us the smaller set $S' = \{s_0, s_1, \ldots, s_{n-2}\}$, and remove s_{n-1} from S_0, giving us S'_0, and also change W to $W' = W - w_{n-1}$. Is it true that S'_0 is now a solution for the knapsack problem with items S', weights $\{w_0, w_1, \ldots, w_{n-2}\}$, values $\{v_0, v_1, \ldots, v_{n-2}\}$, and maximum weight W'? Explain.