**CSCE 340**

**Programming Assignment 2**

Make sure you use proper style when you name variables/constants and align statements. Do not forget to put your name in the top comments of each of your programs. The format of your programs is very important when learning a programming language and bad style may be penalized with up to 30%. **Please use object-oriented programming style and DO NOT use the standard libraries from the book site.**

Microsoft has just hired you as a consultant to help them reduce costs in their customer service department. They digitally record customer service phone calls and store all the sound files on portable disks at the end of each week. Since your marginal cost is proportional to the total number of disks used, your task is to assign the sound files to disks using as few disks as possible. This problem is NP-hard; thus it seems hopeless to find an efficient algorithm for finding the optimal packing. Instead, your goal is to design heuristics that run fast and produce high quality solutions.

We formulate the bin packing problem as follows:  given a set of N file sizes between 0 and 1,000,000 KB (1 GB), find a way to assign them to a minimal number of disks, each of capacity 1 GB. The worst-fit heuristic is a simple rule that considers the file sizes in the order they are presented:  if the sound file won't fit on any disk, create a new disk; otherwise place the file on a disk that has the mostremaining space. For example, this algorithm would put the sizes 700,000, 800,000, 200,000, 150,000, 150,000 onto three disks: {700,000, 200,000}, {800,000, 150,000}, {150,000}. Note that this does not necessarily lead to the best solution since the five files could fit on two disks.

Your main task is to implement the worst-fit heuristic.

**Perspective.** The bin packing problem is a fundamental problem for minimizing the consumption of a scarce resource, usually space. Applications include: packing the data for Internet phone calls into ATM packets, optimizing file storage on removable media, assigning commercials to station breaks, allocating blocks of computer memory, and minimizing the number of identical processors needed to process a set of tasks by a given deadline. In the latter example, the scarce resource is time instead of space. Cloth, paper, and sheet metal manufacturers use a two-dimensional version of the problem to optimally cut pieces of material (according to customer demand) from standard sized rectangular sheets. Shipping companies use a three-dimensional version to optimally pack containers or trucks.

**Disk data type.** First, implement a data type Disk.java that represents a 1GB disk, and contains a list of all of the files it is sorting. This data type should implement the Comparable<Disk> interface so that you can use it with a priority queue.

**Priority queue ADT.** You will need to develop an efficient data structure to support all of the basic operations to implement the heuristic. For the worst-fit heuristic, you will certainly need insertand delete the maximum. The priority queue [MaxPQ.java](https://www.cs.princeton.edu/introalgsds/34pq/MaxPQ.java.html) (posted on Blackboard) is a judicious choice.**Input and output.** Your client program will read in the set of file sizes (guaranteed to be between zero and a million) from standard input. Your program should output the number of disks used by the heuristic and the sum of all file sizes divided by 1 million (a lower bound on the number of disks required). If the number of files is less than 100, you should also print out the disks in decreasing order of remaining space. For each disk, print out its remaining space and its contents (in the order the file sizes were inserted). Optionally, you may also print out a unique id associated with each disk (assigned in the order the disks were created) to aid in debugging.

% java WorstFit < input20.txt

file sizes sum = 6.580996 GB

total disks = 8

5 325754: 347661 326585

0 227744: 420713 351543

7 224009: 383972 392019

4 190811: 324387 484802

6 142051: 340190 263485 254274

3 116563: 347560 204065 331812

2 109806: 396295 230973 262926

1 82266: 311011 286309 320414

**Worst-fit decreasing.** Experienced travelers know that if small items are packed last, they can fit snugly in the odd gaps in nearly filled luggage. This motivates a smarter strategy:  process the items from biggest to smallest. The worst-fit decreasing heuristic is to do worst-fit, but first preprocess the file sizes so that they are in descending order. A modular way to implement this heuristic is to write a separate program IntegerSorter.java that reads in a sequence of integers and prints them out in descending order. Then you can pipe the results through your worst-fit heuristic.

% java IntegerSorter < input20.txt | java WorstFit

Sum of all files = 6.580996 GB

Disks used = 7

1 211686: 396295 392019

0 94485: 484802 420713

6 47762: 262926 254274 230973 204065

5 44188: 324387 320414 311011

3 18470: 347661 347560 286309

4 1413: 340190 331812 326585

2 1000: 383972 351543 263485

**Analysis.** Run your heuristics on a variety of inputs, with N = 100, 1,000, 10,000, 100,000, and 1,000,000. Do a few trials for each value of N using random weights between zero and one million. In your writeup, comment on the relative effectiveness (number of disks used) of the two heuristics (worst-fit and worst-fit decreasing).

**Deliverables.** Your main programs should be named WorstFit.java and IntegerSorter.java. Submit all of the other files needed by your program (e.g., Disk.java) except for MaxPQ.java, which we will provide. Also include a readme.txt file as usual.