

## Assignment objectives

The purpose of this assignment is to implement a *Greedy Algorithm* to solve the job assignment problem that we solved by brute force (exhaustive search) in Lab 1.

## Introduction

For this assignment you will add new methods to the JobAssignmentFinder class to find a job assignment by using a Greedy Algorithm.

This handout describes only the extensions needed for Lab 5. Refer to the Lab 1 handout for information about the job assignment problem.

NOTE: The lab-marking code will assume that **ALL of the Lab 1 functionality is present!** If you do not feel comfortable using your own Lab 1 code, there is a solution posted in the Lab 5 folder.

## New required methods

There are two new required public methods. You may include other private members/methods if needed.

As always, the names and signatures of public methods **must be declared exactly as specified here, including the exact spelling and capitalization of method names.**

<<< THESE METHODS NEVER DO CONSOLE OUTPUT. >>>

**Public method: getGreedyAssignment(), returns ArrayList<Integer>**

No arguments. The returned ArrayList contains a permutation of the numbers from 0..N-1, where NxN is the dimension of the currently loaded benefit matrix. The returned permutation is the job assignment found by the Greedy Algorithm described below.

You may assume that this method will not be called until after readDataFile() has been called.

**Public method: greedyAssignmentTotalValue(), returns int**

No arguments. Returns the total benefit value of the assignment that was most recently found by the Greedy Algorithm (described below) for the currently loaded benefit matrix.

You may assume that this method will not be called until after readDataFile() has been called.

## The Greedy Algorithm

The greedy strategy for your Greedy Algorithm is:

At each iteration, i.e., for each person (indexed from 0 to N-1), choose the job with the largest benefit that has not yet been assigned to someone else.

For example, if the benefit matrix is:

3	2	6	1	2
4	6	8	7	5

4	6	1	9	3
7	9	2	9	2
7	4	9	5	1

Then the greedy algorithm will proceed as follows:

Iteration 0: Person 0 is assigned Job 2 (largest value job in row 0)

Iteration 1: Person 1 is assigned Job 3 (largest value *available* job in row 1)

Iteration 2: Person 2 is assigned Job 1 (largest value *available* job in row 2)

Iteration 3: Person 3 is assigned Job 0 (largest value *available* job in row 3)

Iteration 4: Person 4 is assigned Job 4 (now the only job still available in row 4)

The resulting greedy job assignment is:

[2, 3, 1, 0, 4]

## Data files

The data files use the same format as Lab 1. See Lab 1 handout for details.

NOTE: There are more data files—bigger ones!—available for this lab. The Greedy Algorithm runs *much* faster than exhaustive search.

## Expected results

This table shows the results you should get for the data files we used in Lab 1, and some new data files provided for Lab 5.

Data file	Greedy assignment	Total value of greedy assignment	Reminder of Lab 1 result, just for comparison
data0.txt (Lab 1)	[2, 3, 4, 1, 0]	37	37
data1.txt (Lab 1)	[1, 0, 6, 4, 3, 5, 2]	55	58
data2.txt (Lab 1)	[5, 6, 9, 1, 4, 3, 7, 2, 8, 0]	8277	8658
data3.txt (Lab 1)	[4, 5, 3, 2, 1, 0, 9, 7, 6, 8]	331	335
data4.txt (Lab 1)	[1, 0, 2, 5, 3, 4]	37	42
data5.txt (Lab 1)	[7, 1, 4, 3, 0, 2, 6, 8, 5]	69	74

data11.txt ... NEW!!!	[4, 7, 1, 0, 5, 6, 2, 3, 9, 10, 8]	967	-
data12.txt ... NEW!!!	[2, 1, 8, 3, 5, 6, 7, 9, 10, 11, 4, 0]	973	-
data37.txt ... NEW!!!	[5, 32, 12, 21, 26, 23, 17, 25, 27, 33, 24, 22, 29, 2, 34, 7, 35, 13, 20, 3, 9, 19, 16, 11, 15, 10, 4, 14, 36, 8, 0, 30, 18, 31, 6, 1, 28]	3388	-
data148.txt ... NEW!!!	It's a really big permutation of the numbers [0..147]	12919	-

## Tips

### You'll need a main() method to test your code

All the same notes about main() are applicable here as with previous labs. I probably will never look at it (except see 🤖 problems).

## Submission information

Due date: As shown on Learning Hub.

Submit the following items to the drop box on Learning Hub:

- Your Java source code (JobAssignmentFinder.java)
- Please *do not zip* or otherwise archive your code or your submission. Submit Java files only.
- Please *do not zip* or include your entire IDE project directory.

## Marking information

This lab is worth 20 points. 5 points are reserved for conformance with the COMP 3760 Coding Requirements (handout in Learning Hub “Content” section). **Write your name and ID in your code comments!**

## Virtual donut problems!

NOTE: If you try either of these problems, put some code to demonstrate them in a main() method (can be either in your JobAssignmentFinder class or a driver class) and include it with your submission. **WRITE ME A NOTE WITH YOUR SUBMISSION, SO THAT I WILL KNOW TO TAKE A LOOK AT IT!**

### Problem 1: Try a different greedy strategy


Iterate over the *jobs* (from 0 to  $N-1$ ); for each job, find a *person* who has not yet been assigned any job, and for whom assigning this job will have the largest benefit. Assign the job to that person.

- Run your algorithm on all the data files. How does it perform compared to the other Greedy Algorithm? How about Exhaustive Search?

### Problem 2: Try *another* different greedy strategy

Consider the data file donutdata.txt.

- Find the maximum-valued job assignment with exhaustive search (Lab 1). What is its total value?
- Use the greedy algorithm (the main one from Lab 5) to find a job assignment. What is its total value?
- Compare the above two results. How do they look compared to each other, and how does this differ from the results for all the other data files? How did this happen?
- Invent another greedy strategy that might do a better job (at least with this data file).
- Implement your new greedy strategy, and see how it compares to the other greedy strategy(ies), and to exhaustive search, on all of the sample data files.
- Fill out this table:

Big-O class of getMaxAssignment() function from Lab 1	Big-O class of getGreedyAssignment() function from Lab 5	Big-O class of greedy function from  Problem 2