# Homework 2

MSSC 6000 – Scientific Computing – Spring 2023

Due Monday, February 20, on D2L, by 11:59pm

Before you read the questions, read this important information about using lists, sets, tuples, and dicts in functions. When you pass variables into functions, the function does not receive *a copy* of the variable, it receives the variable itself. So if you modify the variable in the function, it will get modified in your outer code too. For example if you do:

```python
def change_list(L):
        L.pop(0)

L = [1, 2, 3, 4]
print(L)
change_list(L)
print(L)
```

then the result will be

```
[1, 2, 3, 4]
[2, 3, 4]
```

because the function changed the list. This often takes people by surprise, and can be hard to debug! To avoid this behavior, make a copy of the variable at the start of the function and use that. So if we run:

```python
def change_list(L):
        new_list = list(L)
        new_list.pop(0)

L = [1, 2, 3, 4]
print(L)
change_list(L)
print(L)
```

then we will see

```
[1, 2, 3, 4]
[1, 2, 3, 4]
```

You can make copies of sets, tuples, or dicts with `set(S)`, `tuple(T)`, and `dict(D)`.

**Starting with this assignment, your code is expected to have comments explaining how it works.** You don't need a comment for every line – for example, if the line says `L.sort()`, you don't need a comment saying `# sort the list`. Your comments should explain basically what your code is intending to do in each chunk, so I can follow along, and *so I can give partial credit if there are errors*.

Now onto to the questions! There is an attached zip file (on D2L) with all of the data files mentioned.

**Please submit a single document for all of your written answers (pdf, doc, handwritten and scanned, etc). Then submit separate .py files for the python scripts requested.**

1. In this question, you will be given two lists of integers $L_1$ and $L_2$ that have the same length. Your goal is to find the way of pairing up each number of $L_1$ with a number from $L_2$, multiplying each pair together, then adding up the results, that gives you the *smallest* final answer. For example, if your lists are:

$$L_1 = [-2, 5, -1] \qquad L_2 = [3, 1, 2]$$

   one possible way of pairing up the numbers is

$$(-2)(2) + (5)(1) + (-1)(3) = -4 + 5 - 3 = -2$$

   but this is not the best answer because the pairing

$$(-2)(3) + (5)(1) + (-1)(2) = -6 + 5 - 2 = -3$$

   is even smaller.

   For this question you should

   (a) Think up a greedy algorithm to solve it, and describe it to me. (Think it through on paper and in your head before you code anything!)

   (b) Do you think your algorithm is actually optimal? If so, give a short justification why you think this (a formal proof is not necessary). If not, give an example that shows your algorithm is not optimal.

   (c) Implement your algorithm in Python.

   (d) Run your algorithm on the test cases I have provided in the zip file.

   The zip file has a file with a function you can use to read in the data from the files. It returns a list of two lists.

   For this question, you will submit:

   - Your text answers to parts (a) and (b).
   - Your code for part (c). You should have a function called `greedy(list1, list2)`, which I will run on the three test cases to see how good your answer is. The output of that function should be to print the smallest sum of the products of the pairs that you can find. For example,

```
greedy([-2, 5, -1], [3, 1, 2])
-3
```

2. For this question you will implement the three greedy algorithms for the knapsack problem that we discussed in class. As we said at the time, none of them are optimal, so don't expect them to be! If you write your code well, you will be able to use the same main function for all three, with just a small change for the different definitions of "best". (In general, you should try not to have big chunks of duplicate code.) The three greedy algorithms are:

   - "best" = "lightest weight"

- “best” = “highest value”
- “best” = “largest ratio of value to weight”

I have provided test files in the following format. The first line is a floating point number telling you the capacity of the knapsack. Each following line is an item with three parts: the name of the item (a short random string to help me check your output), the weight of the item, and the value of the item. For example, the file

```
40.0
qkx 10.31 10.2
bdn 34.33 40.01
mrm 47.12 77.47
ulx 26.85 69.96
```

represents a knapsack whose capacity is 40, and four possible items with names “qkx”, “bdn”, “mrm”, and “ulx”. The item “qkx” has weight 10.31 and value 10.20.

The output of your code should be to print, for each of the three versions of “best”, the sorted list of names of the items you picked, the total weight of the items you picked, and the total value of the items you picked. For example, your output for the sample file above might look like (in this case they are all the same):

```
lightest
solution: ['qkx', 'ulx']
weight: 37.16
value: 80.16

most valuable
solution: ['qkx', 'ulx']
weight: 37.16
value: 80.16

largest ratio
solution: ['qkx', 'ulx']
weight: 37.16
value: 80.16
```

The zip file has three data files and a file with a function you can use to read in the data from the files. It returns a tuple of two things. The first is the capacity, and the second is a list of triplets of the form (name, weight, value).

For this question, you will submit:

- Just your code, which I will run on the three data files, and which should print output like the example above when I do.

3. As a graduate student, you have a very busy day ahead of you. You have $n$ tasks that you could do. Each one has: an amount of time it takes to do (duration, in minutes), a deadline (given in number of minutes from the start of the day, which we’ll call 0), and an amount of profit you get for completing it (in dollars, or gold stars, or slices of pizza, or whatever we pay graduate students with these days). You can’t do jobs that would finish after the deadline. Your goal is to maximize profit.

   To make sure you understand the problem, here is an example:

| Job # | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Duration | 2 | 5 | 1 | 1 | 2 |
| Deadline | 5 | 8 | 2 | 3 | 2 |
| Profit | 3 | 2 | 1 | 3 | 5 |

Job #1, for example, takes 2 minutes to complete, and its deadline is 5 minutes after the start of the day. So you can start it right away (at minute 0), or at minute 1, 2, or 3, but you can't start it at minute 4, because then you would miss the deadline.

One (non-optimal) solution for this example would be to do jobs 3, 4, and 2, in that order. In this solution, job 3 runs from time 0 to time 1, job 4 runs from time 1 to time 2, and job 2 runs from time 2 to time 7. The total profit is $1 + 3 + 2 = 6$. The optimal solution in this case is to do jobs 5, 4, and 1, in that order. (At least, I think so, I only did it in my head.)

This is a type of **Job Scheduling Problem**, and it models many real-world optimization problems like scheduling tasks to workers and scheduling computer processes. Please do not look up solutions to this problem while completing the assignment!

For the question, your assignment is to

(a) devise a greedy algorithm to solve this problem, and explain it to me in writing in as much detail as necessary so that I could code it myself if I needed to;

(b) invent a sample set of jobs that you think is interesting, and show me how your algorithm selects a subset of the jobs as its solution;

(c) come up with one additional example of a real-world optimization problem that could be modeled as a job scheduling problem (it is okay if the real-world problem has extra constraints and variables).

4. Your local GameStop has just received a new shipment of 60 Playstation 5 consoles, a highly-coveted product. A mob of people have showed up outside, all wanting to buy one. Each person has:

- a dollar amount they are willing to pay for a console,
- a certain number of minutes they are willing to wait before they just leave without one.

GameStop only has one cashier, and each transaction takes exactly one minute. So, it will take them one hour to sell their 60 consoles.

If a person is willing to wait 0 minutes, it means they must be the first person served, or they will leave. If a person is willing to wait 1 minute, it means they must be served first or second, or they will leave. The highest number of minutes a person will ever need to wait is 59, because the last console will be sold 59 minutes after the start.

Your input data is a list of customers containing the amount they are each willing to pay and the number of minutes each of them will wait. Your goal is to choose the combination of customers that will maximize the amount of money GameStop makes.

For example, consider the sample file:

```
159 1
705 2
842 0
269 2
155 0
557 2
```

There are two people who will leave right away if they are not the first person served. One will pay \$842 and the other will pay \$155. The optimal configuration here is \$842 + \$705 + \$557 = \$2104.

For this question you should:

(a) Think up a greedy algorithm to solve it, and describe it to me. (Think it through on paper and in your head before you code anything!) This one is not super obvious, and different people may come up with different solutions!

(b) Implement your algorithm in Python.

(c) Run your algorithm on the test cases I have provided (more details below).

Your algorithm should print a list of the person who will buy a game console at minute *i* for each minute in order, and the total money made on the next line. If no person is buying at minute *i*, display `None` instead. For example, the output for the sample file above would be:

```
[(842, 0), (557, 2), (705, 2), None, None, ...,  None]
2104
```

(I have hidden 54 "None"s to save space.)

The zip file has the three data files I will run your code on, and a function you can use to read in the data. For the easy and medium cases, I will also give you the best value that I was able to find: my best solution for the easy test case is $4332, and my best solution for the medium test case is $49647.

For this question you should submit:

- Your text answers to part (a).
- Your code for part (b). You should have a function called `gamestop(customer_list)`, where `customer_list` is the list of (money, time) pairs that you're reading in from the data. I will run this function on the three test cases to see how good your answer is.