# MATH 182: HOMEWORK 3

Due Wednesday Oct 26 at 12pm.
Submit to "Homework 3 Programming" the following:

- The file(s) containing your code.
- The file `submission_a.txt`
- The file `submission_b.txt`
- The file `submission_c.txt`

## Problem A

Let `optimal_change` be the function which takes as input a positive integer array `A` containing `1`, and a positive integer `n`, and returns a smallest array consisting of elements of `A` (maybe repeated), whose sum is `n`. Note that there could be more than one smallest array; you just have to return one of them.

Apply `optimal_change` to each row of `data_a.txt`, and output each result as a row of `submission_a.txt`.
For instance, if `data_a.txt` was the following:

```
([1, 5, 10, 25], 97)
([1, 5, 10, 25], 40)
([1, 5, 10, 25, 20], 40)
([5, 10, 15, 1], 20)
([1], 1)
([1], 4)
```

Then `submission_a.txt` could be the following:

```
[25, 25, 25, 10, 10, 1, 1]
[25, 10, 5]
[20, 20]
[10, 10]
[1]
[1, 1, 1, 1]
```

## Problem B

Let's say you have a bunch of tasks and you want to allocate them to three people, you want to make sure each person has roughly the same amount of work. The following algorithm is designed for this scenario.

A **tripartition** of a positive integer array is a partition of it into three subsets. Each block has a sum, and the max of these sums is called the **width** of the tripartition. A tripartition is **optimal** if it has the smallest width out of all tripartitions.

Let `optimal_tripartition` be the function which takes as input a positive integer array, and returns an optimal tripartition of it.

Implement `optimal_tripartition` using dynamic programming, apply it to each row of `data_b.txt`, and output each result as a row of `submission_b.txt`.

For instance, if `data_b.txt` was the following:

```
[1, 2, 3]
[5]
[6]
[7, 9]
[4, 5, 6, 7]
[10, 4, 3, 2]
[1, 2, 3, 4, 5, 6]
[]
```

Then `submission_b.txt` could be the following:

```
([1], [2], [3])
([], [5], [])
([], [], [6])
([7], [], [9])
([4, 5], [6], [7])
([10], [4, 3, 2], [])
([1, 6], [2, 5], [4, 3])
([], [], [])
```

## Problem C

Implement the greedy algorithm for the knapsack problem with replacement, where the "greediness" condition is that you keep taking the object with the highest value per weight ratio (this algorithm doesn't give the optimal value), and if there's a tie for higher ratio, take the object with the highest value.

Call this function `greedy_knapsack`. The input is an array of pairs `(value, weight)`, where both value and weight are positive integers, and a positive integer `W`. Apply `greedy_knapsack` to each row of `data_c.txt`, and output each result as a row of `submission_c.txt`.

For instance, if `data_c.txt` was the following:

```
([(5, 2), (6, 3), (7, 4)], 7)
([(1, 1), (6, 2)], 5)
([(20, 9), (8, 8)], 3)
```

Then `submission_c.txt` could be the following:

```
[(5, 2), (5, 2), (5, 2)]
[(6, 2), (6, 2), (1, 1)]
[]
```