Daniel Yoo (26623875)

We decided to represent our gene as a binary number of length 12 stored in an array. 1 representing investing in the project and 0 representing not investing in the project.

This array [0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1] would represent the following:

First four indexes represent projects of year 1
Next four indexes represent projects of year 2
Last four indexes represent projects of year 3

Therefore, [0, 1, 1, 0 ...] means invest in project 2 and 3 of year 1.

To generate the children, we decided to randomly pick 2 parents and add first half of the first parent to the last half of the second parent. When picking a parent, there was a random chance of picking the same parent twice (it breeds with itself).

The mutation function looped through each gene in the population and mutate the gene when it would randomize and get a hit. The mutation would switch a single binary in the gene into it's opposite.

For the fit function, we decided to return 0 for all genes that had any sort of cost that exceeded the allowed budget even if for only a single year. If not, we decided to return the total return on investment amount of that gene.

To select the parents for the next generation, we sorted the population with the highest fitness value being first. We took the first n parents and filled the rest of the population with offsprings of these parents and randomly mutated some of the population given a percentage chance.

The algorithm would keep creating new generations until it reached the maximum allowed generation defined by us.

These were the parameters for our genetic algorithm:

```
population_size = 10
generation_max = 50
gene_size = 12
mutate_chance = 0.2
parent_count = 5
```

The population size represents how many genes in a single generation.
The generation max represents how many generations to go through before terminating.
The gene size represents the length of the gene represented by binary.
The mutate chance is the chance that a single gene will mutate.

The parent count is how many genes from a generation will become parents for the next generation.
These are example outputs for a single generation of our genetic algorithm:

```
Generation 1
(1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0) , fitness: 1.90
(0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1) , fitness: 1.60
(0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0) , fitness: 1.60
(1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1) , fitness: 1.50
(1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0) , fitness: 0.90
(0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0) , fitness: 0.30
(1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1) , fitness: 0.00
(1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1) , fitness: 0.00
(1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0) , fitness: 0.00
(0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0) , fitness: 0.00
Selected parents :
(1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0)
(0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1)
(0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0)
(1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1)
(1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0)
--------------------
Generated children :
(1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1)
(0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1)
(1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0)
(1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1)
(1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1)
--------------------
```

The first section represents the gene representation of our population with it's fitness value.
After, we display the parents that were selected to breed.
The last section shows us the children that were generated by random.

From running our algorithm, we realized that it would sometimes not find the optimal solution which should be a fitness value of 2.4. I think this is due to the randomness of a genetic algorithm. However, this may be fixed by increasing our generation size in exchange for computing time and power. Other than that, increasing the population size and mutation chance could increase the likelihood of finding the most optimum solution by randomizing some of the genes more aggressively, which helps promote alternatives.

Given all this, the final optimized solution should be represented by this gene:

```
(1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1) , fitness: 2.50
```