Indroduction to algorithms

*Note: this is the first page of my notes for algorithms base.

Q: *What is an algorithm?*

A: Algorithm is a set of instructions that has to be executed to solve a certain problem.

! Pseudocode is used to declare a solution to a certain problem in a code-like view. It is used to make an algorithm more undrestandable.

Here are some of the main things you have to know about pseudocode:

1. It does not require strict gode syntax and ignores syntax-connected problems you could have when writing actual code.

2. It is not completely detached from the actual code: actually, it still looks like normal code and can use main structural concepts of a certain PL. Here is a quote from Wiki:

   > *Pseudocode often uses structural conventions of a normal programming language, but is intended for human reading rather than machine reading It typically omits details that are essential for machine understanding of the algorithm, such as variable declarations and language-specific code. The programming language is augmented with natural language description details, where convenient, or with compact mathematical notation.*

3. There is no ***strict*** standart for pseudocode. Though, there are still certain standarts and common practices for it.

4. You have to write pseudocode **in such a way, so every other programmer could understand what you have written, even if they do not know your main programming language.**

We will use a common task from our daily life: "exchange"

Given a certain amount of money (we will name it `money`). The task is to exchange given money to the least possible amount of coins (`c1, c2, c3, ..., c(d)`) that are stored in an array `c`. The amount of all the possible nominals is `d`.

Here is the easiest way to solve this problem:

```
Change(money, c, d):
    while money > 0:
        coin = ... // coin with the highest nominal (nominal <= money)
        money = money - coin
```

Correct and incorrect algorithms

! An algortihm is called correct when it gives a correct answer to every possible set of input data. Otherwise, the algorithm is incorrect.

If we take a closer look at our algorithm, we will notice that it is incorrect:

For fiven amount of money = 40 and array of coins = (25, 20, 10, 5, 1) correct answer would be `20x2`. But, according to the algorithm, the answer is `25x1 + 10x1 + 5x1`. The algorithm is incorrect.

To correct our alorithm, we have to look through all the possible combinations of coins tha wolud give given amount of money when added and then output the one where least amount of coins used. Here is correct pseudocode equivalent of our algortihm:

```
// npte: ∑ symbol means sum of numbers from i to m.
BruteForceChange(money, c, d):
    smallestNumberOfCoins = 0
    for each combinations of coins (i_1,...,i_d)
    // от (0,...,0) до (money/c[1],...,money/c[d])
        valueOfCoins = ∑ i_k*c_k // сумма по всем k от 1 до d
        if valueOfCoins = M:
            numberOfCoins = ∑ i_k // суммарное количество монет
            if numberOfCoins < smallestNumberOfCoins:
                smallestNumberOfCoins = numberOfCoins
```

```
                change = (i_1, i_2, ... ,i_d)
    return change
```

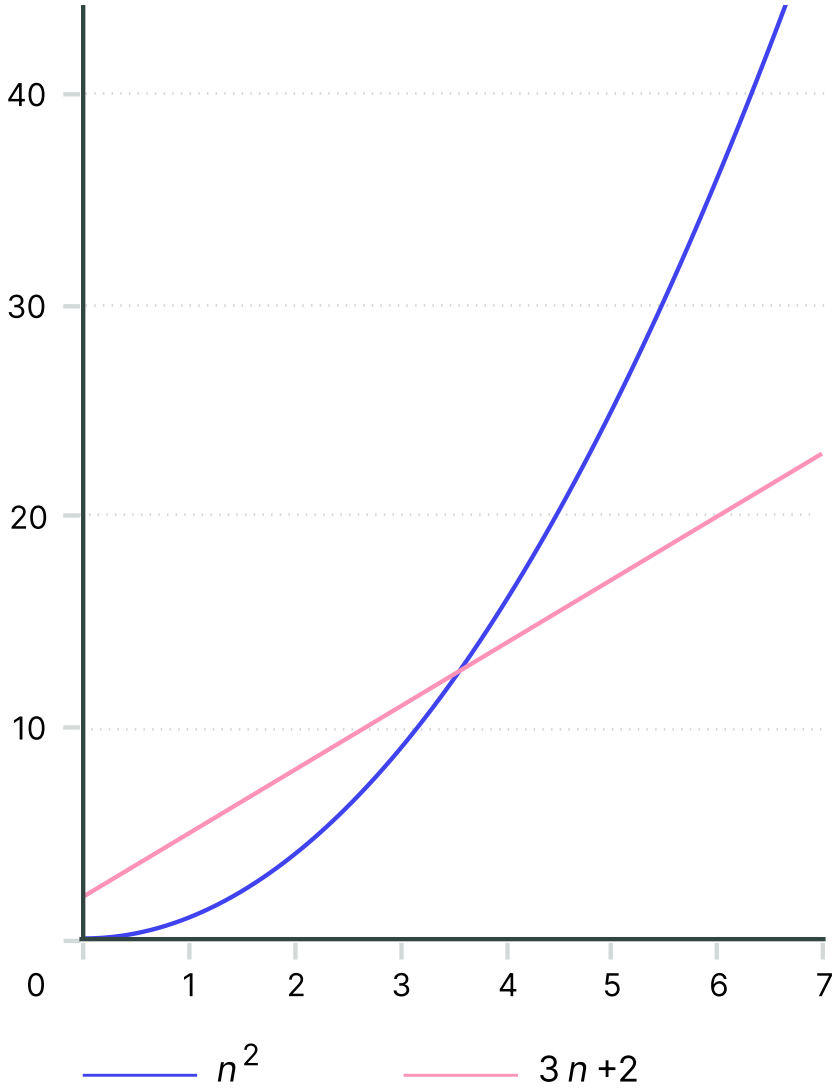
`change` variable cannot be an unoptimal amount of coins because it goes through *every single combination* and will find the optimal solution at one moment. Every single combiation will have atleast the same amount ofused coins as the optimal one, so unoptimal solutin cannot be returned.

Speed is the key.

Obviously, you do not want your program to be executing for a thousand years. This is where the speed of an algorithm plays a great role. To get things straight: recently I was solving a task and wrote a huge pile of code that actually worked correctly, but it did not fit the time limit. I failed to do that task because of time limits.

If we take a look at our algorithm and pseudocode we can notice that `for` loop will be repeated for `money^d` amount of times.

This is where math starts to kick in: if we have an algortihm A that will be executed with speed of `n^2` and an algorithm B that will be executed with speed of `3n+2` we may think that the first algorithm would be faster (it is actually faster, but only on small values of variable `n`). In fact, algorithm B would be faster. If we take a look and compare graphs of these 2 algorithms, we will notice that execution time of algorithm B is lesser in most of the cases:



Algorithms like A are named exponential algorithms. Algorithms like B are linear, just like the name of this graph. Algorithms woth execution spedd of `n^2, n^3, etc.` are named polynominal. Exponential algorithms have max execution speed of `n^k`, wherer `k` variable is constant that is not connected with other input data.

Obviously, the lower an algorithm's execution speed, the less practical this algorithm is.