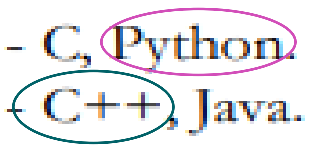
ELİF BAYIR

[illegible]

REPORT

First of all, I used my language choice in this direction.



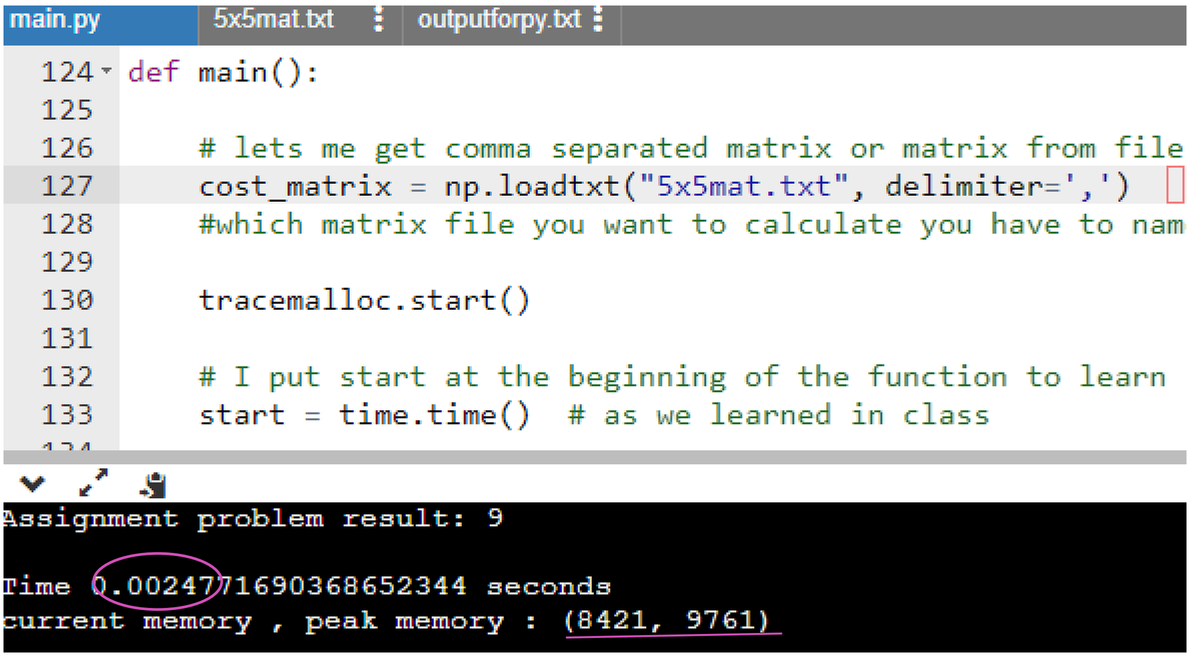
- C, Python.
- C++, Java.

I am trying our 5x5,10x10,100x100 and 1000x1000 matrices for my python code..

Let's look at the change in time and memory usage versus matrix size;

For 5x5 matrix;

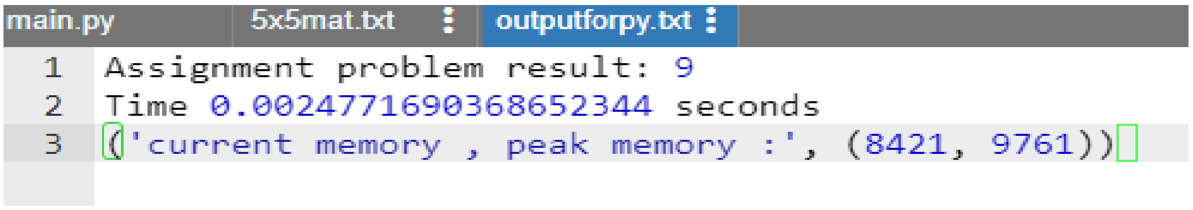
(I will use the same matrix for C++)



```
def main():

    # lets me get comma separated matrix or matrix from file
    cost_matrix = np.loadtxt("5x5mat.txt", delimiter=',')
    #which matrix file you want to calculate you have to nam

    tracemalloc.start()

    # I put start at the beginning of the function to learn
    start = time.time()  # as we learned in class
```

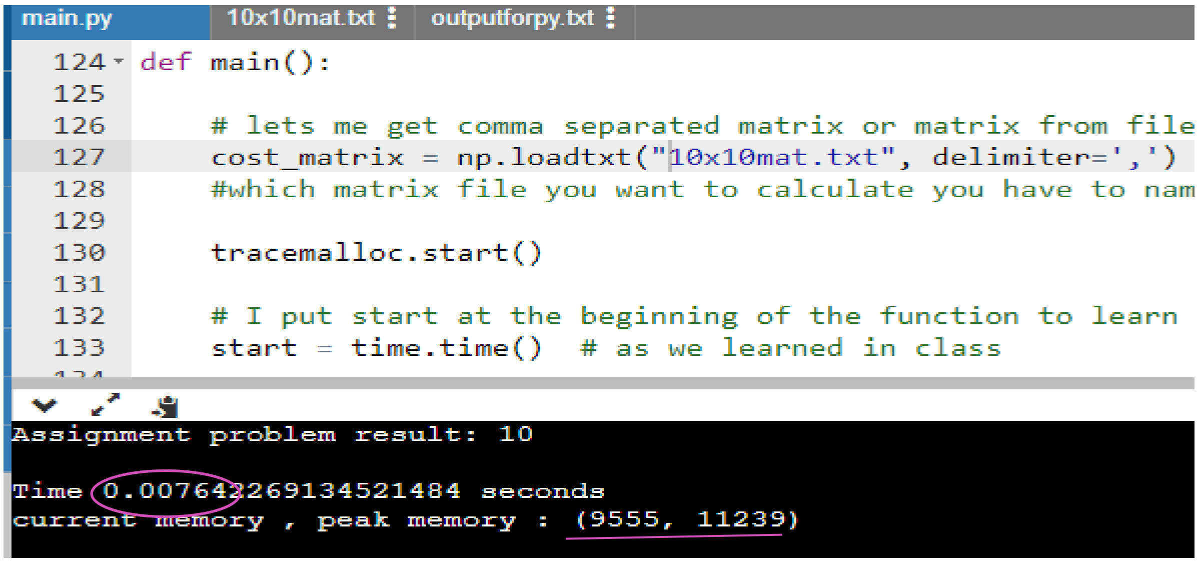
```
Assignment problem result:  9

Time 0.0024771690368652344 seconds
current memory , peak memory :  (8421, 9761)
```

and ı'm putting these results in my file(outputforpy.txt).



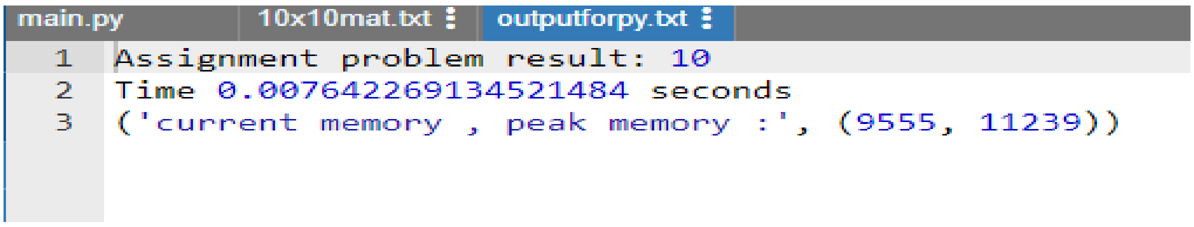
```
Assignment problem result: 9
Time 0.0024771690368652344 seconds
('current memory , peak memory :', (8421, 9761))
```

For 10x10 matrix;



```
def main():

    # lets me get comma separated matrix or matrix from file
    cost_matrix = np.loadtxt("10x10mat.txt", delimiter=',')
    #which matrix file you want to calculate you have to nam

    tracemalloc.start()

    # I put start at the beginning of the function to learn
    start = time.time()   # as we learned in class
```

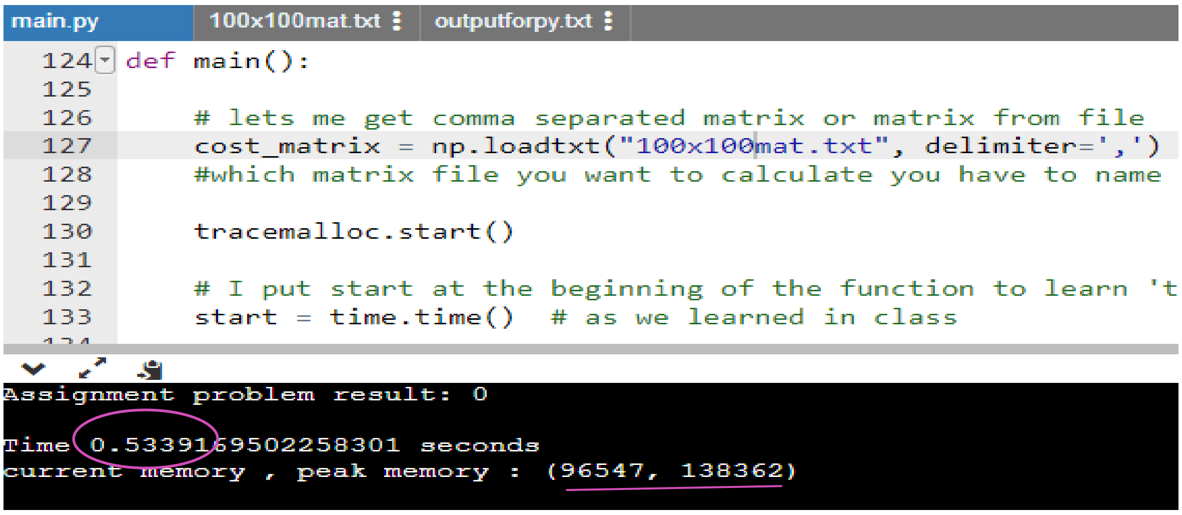
```
Assignment problem result: 10

Time 0.007642269134521484 seconds
current memory , peak memory : (9555, 11239)
```



```
Assignment problem result: 10
Time 0.007642269134521484 seconds
('current memory , peak memory :', (9555, 11239))
```

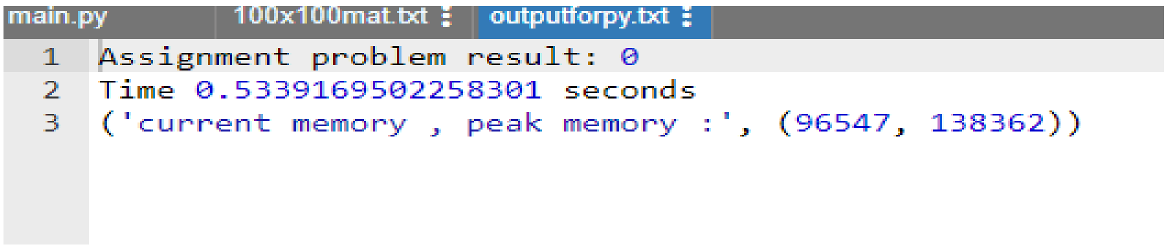
There is a slight change in time and memory.

For 100x100 matrix;



```
def main():

    # lets me get comma separated matrix or matrix from file
    cost_matrix = np.loadtxt("100x100mat.txt", delimiter=',')
    #which matrix file you want to calculate you have to name

    tracemalloc.start()

    # I put start at the beginning of the function to learn 't
    start = time.time()   # as we learned in class
```

```
Assignment problem result: 0

Time 0.5339169502258301 seconds
current memory , peak memory : (96547, 138362)
```

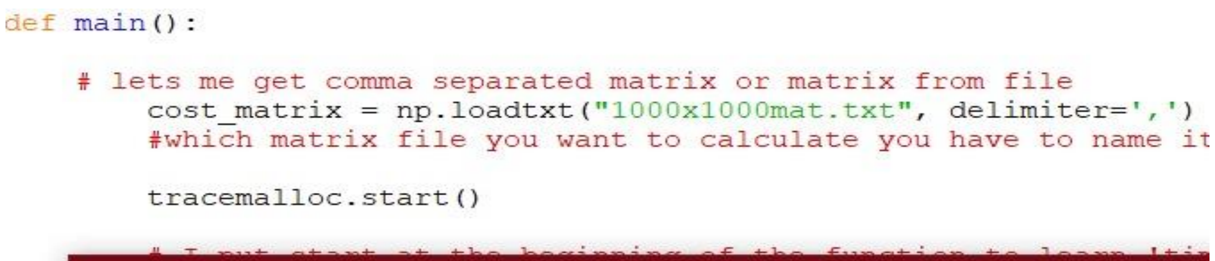

main.py | 100x100mat.txt | outputforpy.txt

```
Assignment problem result: 0
Time 0.5339169502258301 seconds
('current memory , peak memory :', (96547, 138362))
```

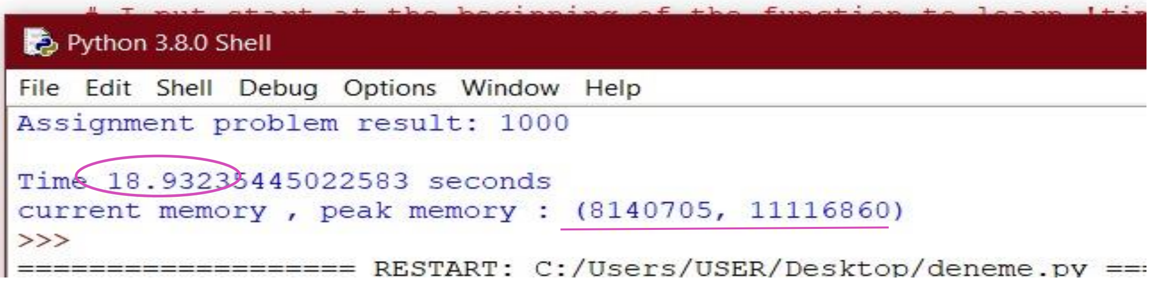
As you can see, the change has now increased by about 100 times per second at a noticeable level, and memory usage has increased by about 10 times.

The reason for this increase is, of course, the growth of my matrix The calculation time of the code increases in direct proportion to the growth of our matrix.In the same way, my memory usage increases with the growth of the matrix, my current memory and peak memory usage increases.

For 1000x1000 matrix;

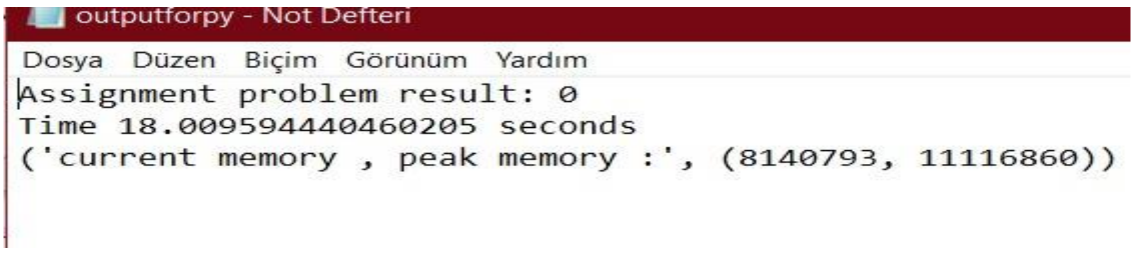


```
def main():

    # lets me get comma separated matrix or matrix from file
        cost_matrix = np.loadtxt("1000x1000mat.txt", delimiter=',')
        #which matrix file you want to calculate you have to name it

        tracemalloc.start()
```


Python 3.8.0 Shell

File Edit Shell Debug Options Window Help

```
Assignment problem result: 1000

Time 18.93235445022583 seconds
current memory , peak memory : (8140705, 11116860)
>>>
=================== RESTART: C:/Users/USER/Desktop/deneme.py ==
```


outputforpy - Not Defteri

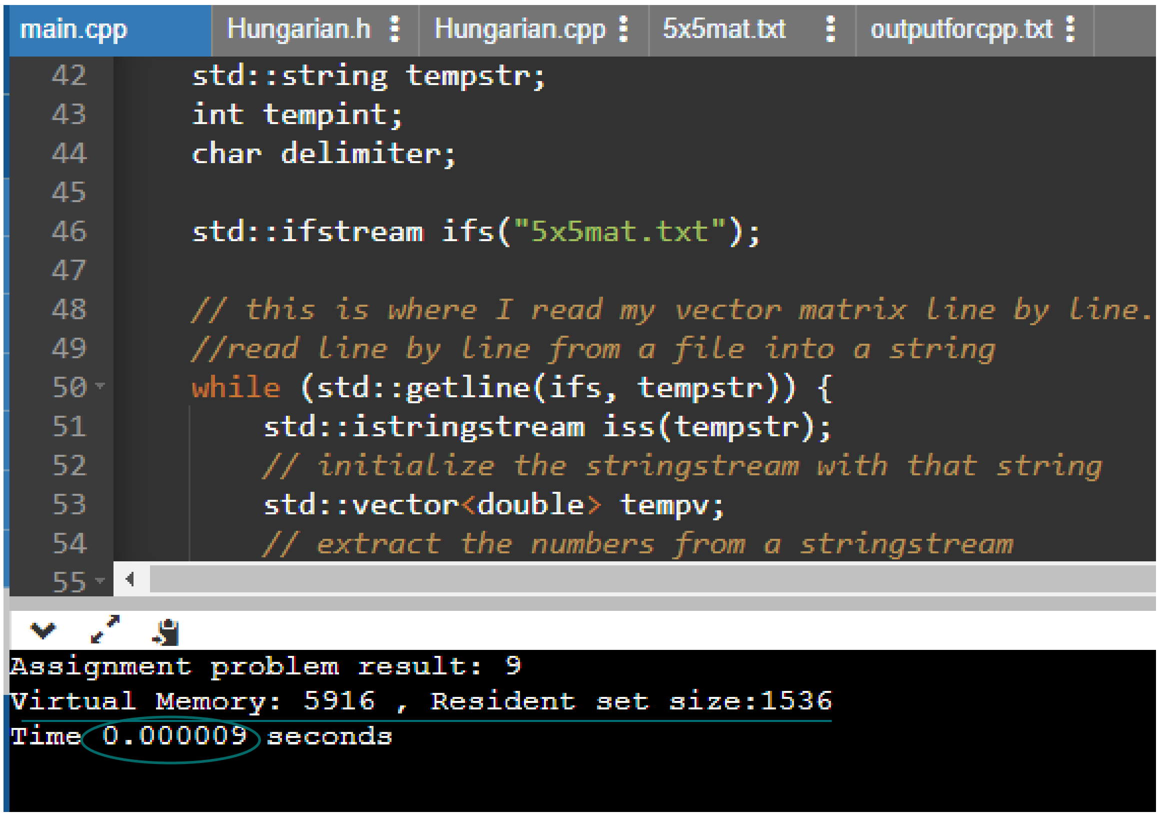
Dosya Düzen Biçim Görünüm Yardım

```
Assignment problem result: 0
Time 18.009594440460205 seconds
('current memory , peak memory :', (8140793, 11116860))
```

The increase was obvious.I can't work on larger matrices because my computer gave me error terminating the experience in a 1000x1000 matrix.

I'm using the same datasets of mine to make a better comparison.

For 5x5 matrix;

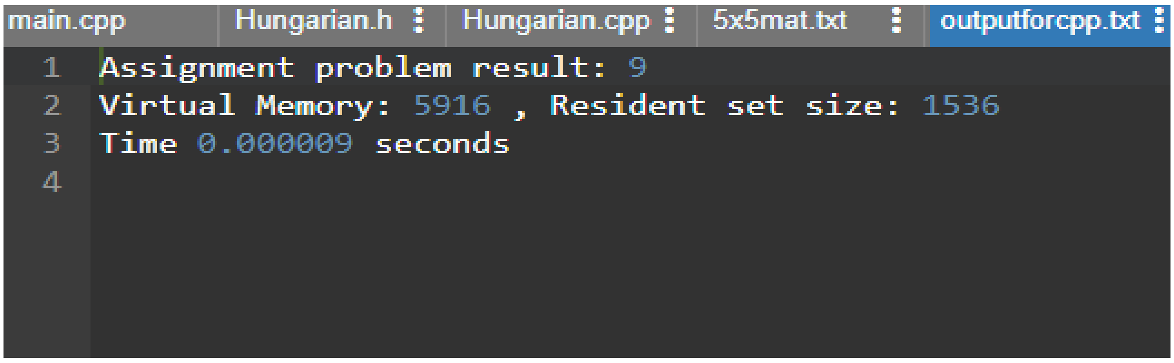


```
    std::string tempstr;
    int tempint;
    char delimiter;

    std::ifstream ifs("5x5mat.txt");

    // this is where I read my vector matrix line by line.
    //read line by line from a file into a string
    while (std::getline(ifs, tempstr)) {
        std::istringstream iss(tempstr);
        // initialize the stringstream with that string
        std::vector<double> tempv;
        // extract the numbers from a stringstream
```

```
Assignment problem result: 9
Virtual Memory: 5916 , Resident set size:1536
Time 0.000009 seconds
```

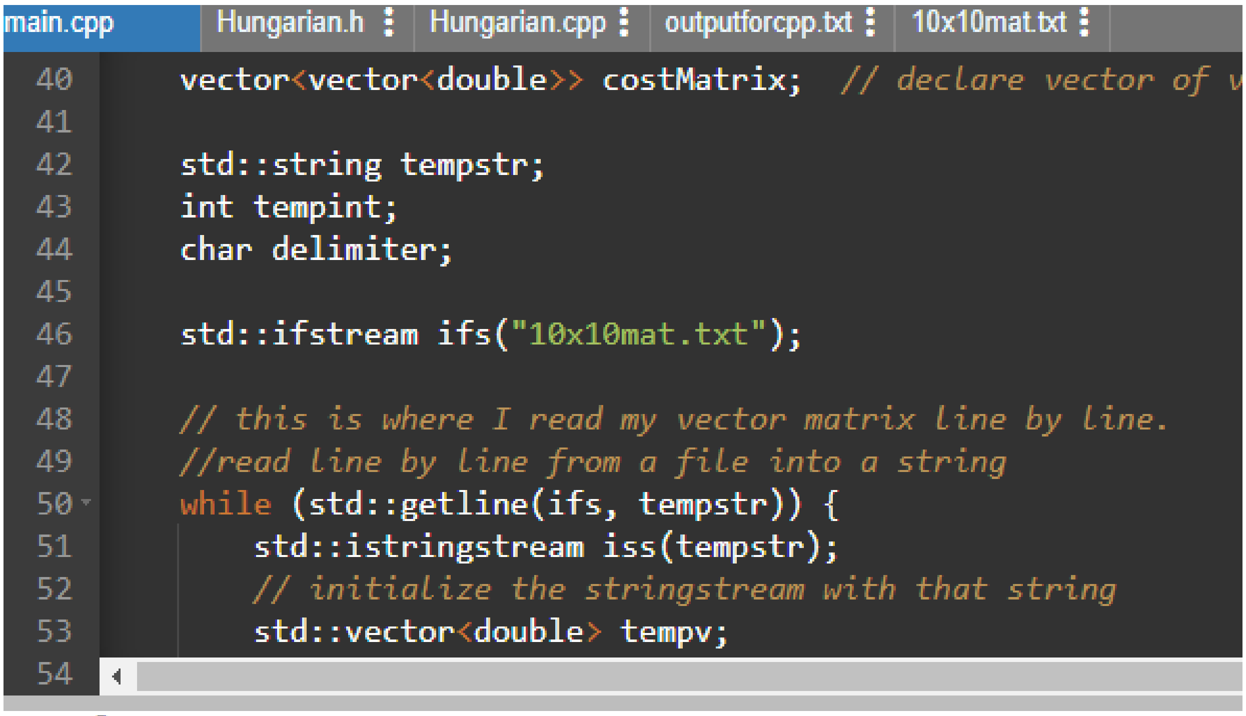
I used the same 5x5 matrix and found the result of 9 in python.I noticed that it does it in less time than in python.

I would like to see my experiment results in an output file,

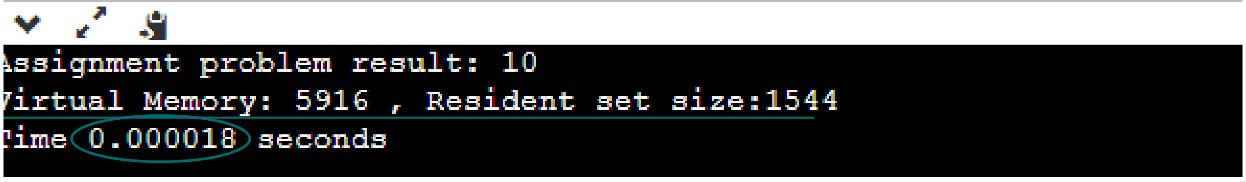


```
Assignment problem result: 9
Virtual Memory: 5916 , Resident set size: 1536
Time 0.000009 seconds
```

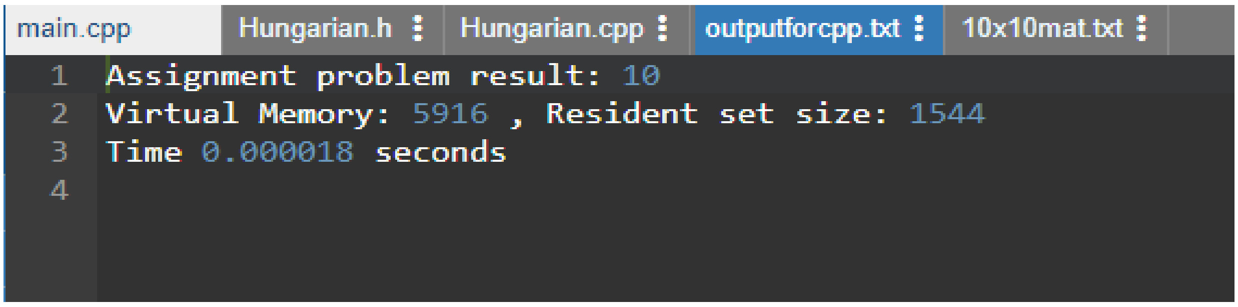
For 10x10 matrix;



```
    vector<vector<double>> costMatrix;  // declare vector of v

    std::string tempstr;
    int tempint;
    char delimiter;

    std::ifstream ifs("10x10mat.txt");

    // this is where I read my vector matrix line by line.
    //read line by line from a file into a string
    while (std::getline(ifs, tempstr)) {
        std::istringstream iss(tempstr);
        // initialize the stringstream with that string
        std::vector<double> tempv;

```



```
ssignment problem result: 10
irtual Memory: 5916 , Resident set size:1544
ime 0.000018 seconds
```



```
Assignment problem result: 10
Virtual Memory: 5916 , Resident set size: 1544
Time 0.000018 seconds

```

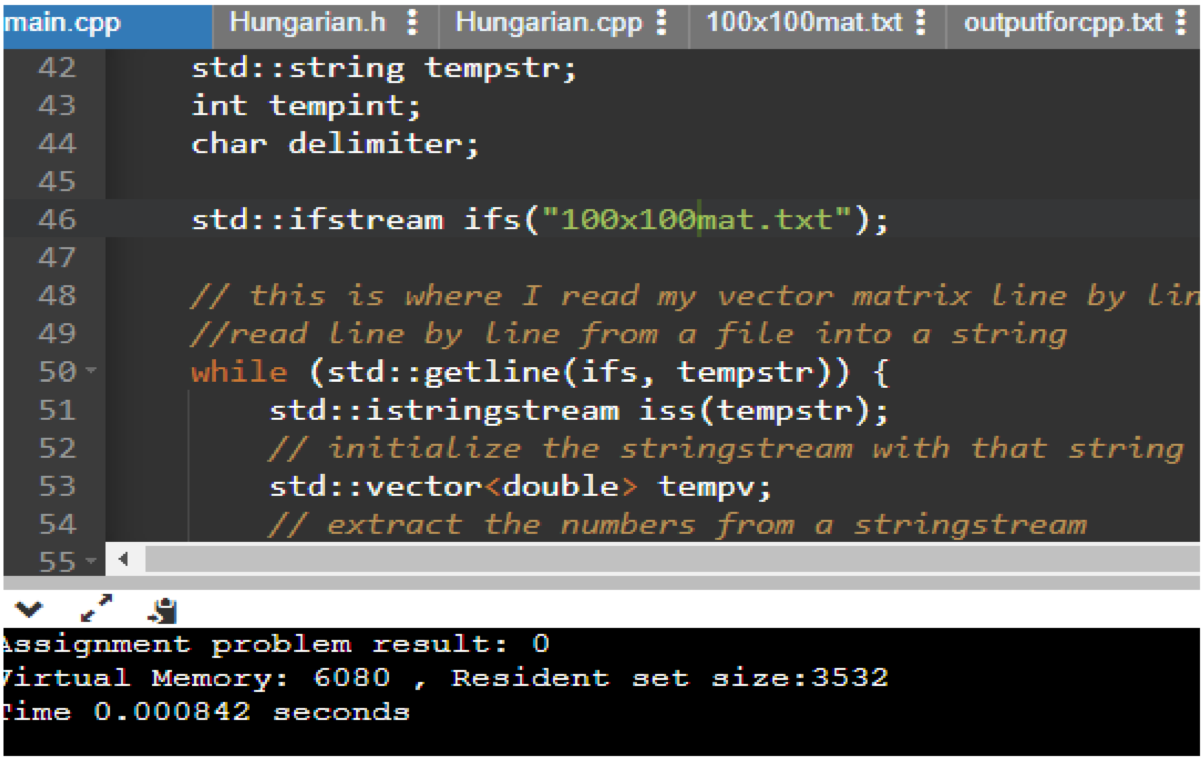
There was a slight difference because the difference between the matrix size is only 2 decimals.

So it turned out about 2 times the time.

5x5 takes 0.000009 second.

10x10 takes 0.000018 second.

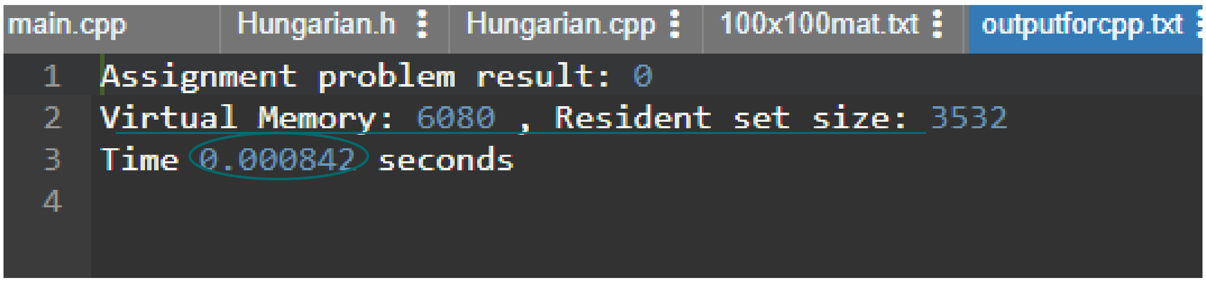
For 100x100 matrix;



main.cpp | Hungarian.h | Hungarian.cpp | 100x100mat.txt | outputforcpp.txt

```
    std::string tempstr;
    int tempint;
    char delimiter;

    std::ifstream ifs("100x100mat.txt");

    // this is where I read my vector matrix line by lin
    //read line by line from a file into a string
    while (std::getline(ifs, tempstr)) {
        std::istringstream iss(tempstr);
        // initialize the stringstream with that string
        std::vector<double> tempv;
        // extract the numbers from a stringstream
```

```
Assignment problem result: 0
Virtual Memory: 6080 , Resident set size:3532
Time 0.000842 seconds
```



main.cpp | Hungarian.h | Hungarian.cpp | 100x100mat.txt | outputforcpp.txt

```
Assignment problem result: 0
Virtual Memory: 6080 , Resident set size: 3532
Time 0.000842 seconds
```

In terms of time, C++ was faster than python, and the increments were smaller in size.

But in the same way, when the matrix grew, the use of time and memory increased in itself.

Unfortunately, I can't produce larger matrices.

I generate my matrices from this site if you want to try it out;

https://catonmat.net/tools/generate-random-matrices

Compare Procedural vs Object-Oriented Programming

Procedural programming focuses on the process and functions.For example, python code was made with functions a separate function was written for the calculation, a separate function was written for the Hungarian algorithm.

OOP focuses on the data and classes.For example, The C++code was made with classes.We called the class 'HungarianAlgorithm'.And In this paradigm, it is easy to maintain code and modify existing code,because it is divided by classes.

Procedural programming languages are not as faster as object-oriented. I realized this when we did it.Because 100x100 is a matrix that C++ does in 0.00842 seconds, while python does in 0.5331 seconds.

Object-oriented and procedural are high-level programming paradigms to solve problems in less time by writing modular code. But OOP is best when it comes to bigger applications as procedural is not good for complex applications.

Object-oriented programming is more secure than procedural programming, because of the level of abstraction or we can say data hiding property. It limits the access of data to the member functions of the same class.

	C++	**Python**
5x5	Time:0.000009 Virtual Memory: 5916 Resident set size:1536	Time:0.0024 Current Memory: 8421 Peak Memory:9761
10x10	Time:0.000018 Virtual Memory: 5916 Resident set size:1544	Time:0.0076 Current Memory: 9555 Peak Memory:11239
100x100	Time:0.000842 Virtual Memory: 6080 Resident set size:3640	Time:0.5331 Current Memory: 96547 Peak Memory:138362
1000x1000		Time:18.93 Current Memory: 8140705 Peak Memory:11116860