هر قسمت از پروژه در یک فایل ipynb (jupyter notebook) گردآوری شده.
داده ها (زمان های اجرای هر الگوریتم) و نمودار ها و جدول های حاصل از آن ها به صورت کاملا اتوماتیک توسط کد تولید میشود و تنها بایستی ورودی های مورد نظر را به آن پاس کنیم.

در هر pdf (که درواقع pdf فایل های جوپیتر است) کد های auto generate و نمودار ها و جداول حاصل از آن قسمت آورده شده است. برای نشان دادن نتایج حاصل از اجرای برنامه توسط اینجانب قسمت هایی با عنوان (Code with saved datas) نیز آورده شده تا نتایج را با times ها از قبل پیدا شده نشان دهد. (همچنین برای راحتی شما این فایل ها در انتهای همین pdf نیز درج شده اند)

در کد های auto generate چهار فانکشن با اسم های functio**ni** (1<=i<=4) وجود دارد که هر یک پیدا سازی الگوریتم i ام هستند. در تابع های دیگر مسائل مربوط به رسم نمودار ها، جداول و محسابه تایم ها پیدا سازی شده است.

تحلیل قسمت الف:
در این قسمت همانطور که از نمودار ها پیداست رشد الگوریتم اول به شدت بیشتر از دیگر الگوریتم هاست. همچنین الگوریتم دوم نیز به نسبت رشد بسیار بیشتری از الگوریتم های سوم و چهارم دارد. و در نهایت از نمودار سوم پی میبریم که الگوریتم چهارم از دیگر الگوریتم ها بهینه تر بوده و رشد زمانی کمتری داشته است.
(نمودار ها و جداول با جزئیات بیشتر نیز در فایل ها موجود و قابل مشاهده است)

تحلیل قسمت ب و ج:
توضیحات مانند قسمت الف است با این تفاوت که زمان ها در اکثر مواقع بیشتر از حالت الف هستند (در قسمت ب به خاطر غیر بهینه بودن و انجام عملیات های نامفید

و همچنین تبدیل boolean به int و در قسمت ج به علت اجرای برنامه های سنگین بر روی سیستم عامل و شلوغ شدن cpu و ram و ایجاد صف در thread های cpu محاسبه الگوریتم ها بیشتر طول کشیده است). همچنین پی میبریم سیر رشد توابع زمان اجرای الگوریتم ها در هر سه بخش بسیار شبیه به هم بوده است.

قسمت د (مقایسه کلی بین قسمت های الف ب ج):

در این قسمت به نتایج جالبی میرسیم. برای مثال در نمودار اول و سوم، الگوریتم ها در حالت ج کند تر عمل کرده اند (که این میتواند به خاطر ضعیف بودن سیستم تست شده باشد!). در حالی که الگوریتم دوم در هر سه وضعیت زمان نسبتا یکسانی را از خود ثبت کرده است (در نهایت در حالت ب کند تر عمل کرده است). در چهارمین الگوریتم نیز اتفاق جالبی افتاده است و آن این است که تاثیر غیر بهینه کردن آن بسیار بیشتر از کند کردن سیستم عامل بوده است.

Code with auto data, table and graph generate:

```
import math
import random
import time
import matplotlib.pyplot as plt
import numpy as np

def solution1(n):
    count = 0
    for i in range(2, n + 1):
        isPrime = True
        for j in range(2, i):
            if(i%j == 0):
                isPrime = False
        if(isPrime):
            count += 1
    return count

def solution2(n):
    count = 0
    for i in range(2, n + 1):
        isPrime = True
        for j in range(2, math.floor(math.sqrt(i))+1):
            if(i%j == 0):
                isPrime = False
        if(isPrime):
            count += 1
    return count

def solution3(n):
    # hasDivisor means divisors except 1 and the number itself
    hasDivisor = [False for i in range(n+1)] # from 0 to n
    for i in range(2, n + 1):
        for j in range(2 * i, n + 1, i):
            hasDivisor[j] = True
    return n - hasDivisor.count(True) - 1 # -1 is for 1

def solution4(n):
    # hasDivisor means divisors except 1 and the number itself
    hasDivisor = [False for i in range(n+1)] # from 0 to n
    for i in range(2, n + 1):
        if(not hasDivisor[i]):
            for j in range(i * i, n + 1, i):
                hasDivisor[j] = True
    return n - hasDivisor.count(True) - 1  #-1 is for 1

def drawGraph(x, y):
```

```
    plt.plot(x,y)
    plt.show()

def drawSummaryGraphs():
    for k in range(0, 3):
        for i in range(k, 4):
            plt.plot(inputs, times[i], label = "solution
{}".format(i+1))
        plt.legend()
        plt.show()

def drawTable(rows, columns, data, title):
    fig, ax = plt.subplots()
    ax.set_axis_off()
    rcolors = plt.cm.BuPu(np.full(len(rows), 0.1))
    ccolors = plt.cm.BuPu(np.full(len(columns), 0.1))
    table = ax.table(
        cellText = data,
        rowLabels = rows,
        colLabels = columns,
        rowColours = rcolors,
        colColours = ccolors,
        cellLoc ='center',
        loc ='upper left')

#     table.auto_set_font_size(False)
    table.set_fontsize(30)
    table.scale(2, 5)
    ax.set_title(title,
                 fontweight ="bold", fontdict={'fontsize': 30})

    plt.show()

def drawAllSingleTG():
    for i in range(4):
        drawTable(["Time"], inputs, [times[i]], 'Solution
{}'.format(i+1))
        drawGraph(inputs, times[i])

def drawStuff():
    drawTable(["Solution 1", "Solution 2", "Solution 3", "Solution
4"], inputs, times, 'Summary')
    drawSummaryGraphs()
    drawAllSingleTG()
    print("="*10 + " Times List " + "="*10)
    print(times)

def calculateTimes():
    for n in inputs:
        start_time = time.time()
```

```
        solution1(n)
        times[0].append(time.time() - start_time)

        start_time = time.time()
        solution2(n)
        times[1].append(time.time() - start_time)

        start_time = time.time()
        solution3(n)
        times[2].append(time.time() - start_time)

        start_time = time.time()
        solution4(n)
        times[3].append(time.time() - start_time)



inputs = [5, 10, 50, 100, 500, 10**3, 5 * 10**3, 10**4, 5 * 10**4,
10**5, 5 * 10**5, 10**6]
times = [[], [], [], []]
outputs = [[], [], [], []]
calculateTimes()
drawStuff()
```

Code with saved datas:

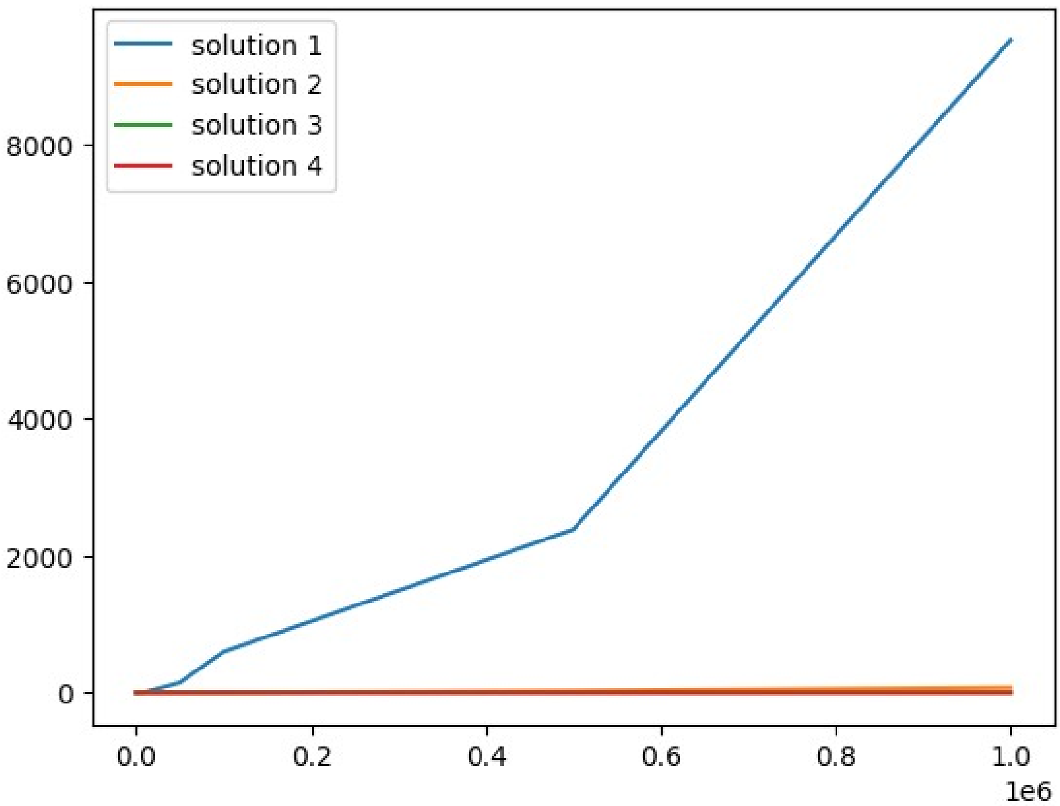
```
import matplotlib.pyplot as plt
import numpy as np

inputs = [5, 10, 50, 100, 500, 10**3, 5 * 10**3, 10**4, 5 * 10**4,
10**5, 5 * 10**5, 10**6]

times = [[8.58306884765625e-06, 8.344650268554688e-06,
0.0001227855682373047, 0.00045990943908691406, 0.01203608512878418,
0.05643773078918457, 1.5574758052825928, 5.795194864273071,
147.73878645896912, 595.1502323150635, 595.1502323150635*4,
595.1502323150635*16], [1.2874603271484375e-05, 9.5367431640625e-06,
5.221366882324219e-05, 0.00012230873107910156, 0.0009887218475341797,
0.002524852752685547, 0.02401137351989746, 0.06473803520202637,
0.7862980365753174, 2.0648396015167236, 26.445839405059814,
72.89753484725952], [8.106231689453125e-06, 6.9141387939453125e-06,
2.5272369384765625e-05, 5.030632019042969e-05, 0.0002853870391845703,
0.0006022453308105469, 0.0033957958221435547, 0.008321762084960938,
0.04799675941467285, 0.08408665657043457, 0.47100043296813965,
1.0639145374298096], [4.76837158203125e-06, 5.4836273193359375e-06,
1.5974044799804688e-05, 2.8371810913085938e-05,
0.00014400482177734375, 0.0002930164337158203, 0.0015158653259277344,
0.0030672550201416016, 0.015823841094970703, 0.03692960739135742,
```

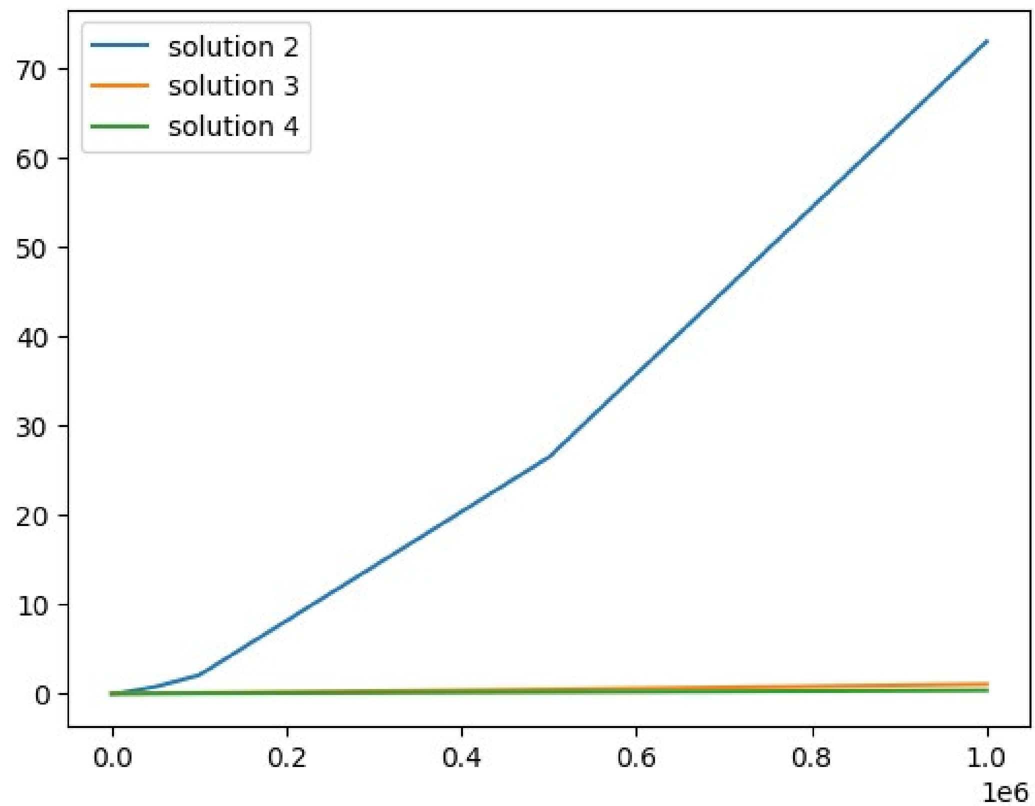
```
0.17017102241516113, 0.360424280166626]]
for i in range(len(times)):
    for j in range(len(times[i])):
        times[i][j] = float("{:.5f}".format(times[i][j]))

def drawGraph(x, y):
    plt.plot(x,y)
    plt.show()

def drawSummaryGraphs():
    for k in range(0, 3):
        for i in range(k, 4):
            plt.plot(inputs, times[i], label = "solution
{}".format(i+1))
        plt.legend()
        plt.show()

def drawTable(rows, columns, data, title):
    fig, ax = plt.subplots()
    ax.set_axis_off()
    rcolors = plt.cm.BuPu(np.full(len(rows), 0.1))
    ccolors = plt.cm.BuPu(np.full(len(columns), 0.1))
    table = ax.table(
        cellText = data,
        rowLabels = rows,
        colLabels = columns,
        rowColours = rcolors,
        colColours = ccolors,
        cellLoc ='center',
        loc ='upper left')

#     table.auto_set_font_size(False)
    table.set_fontsize(30)
    table.scale(2, 5)
    ax.set_title(title,
                 fontweight ="bold", fontdict={'fontsize': 30})

    plt.show()

def drawAllSingleGraphs():
    for i in range(4):
        drawTable(["Time"], inputs, [times[i]], 'Solution
{}'.format(i+1))
        drawGraph(inputs, times[i])

def drawStuff():
    drawTable(["Solution 1", "Solution 2", "Solution 3", "Solution
4"], inputs, times, 'Summary')
    drawSummaryGraphs()
    drawAllSingleGraphs()
```

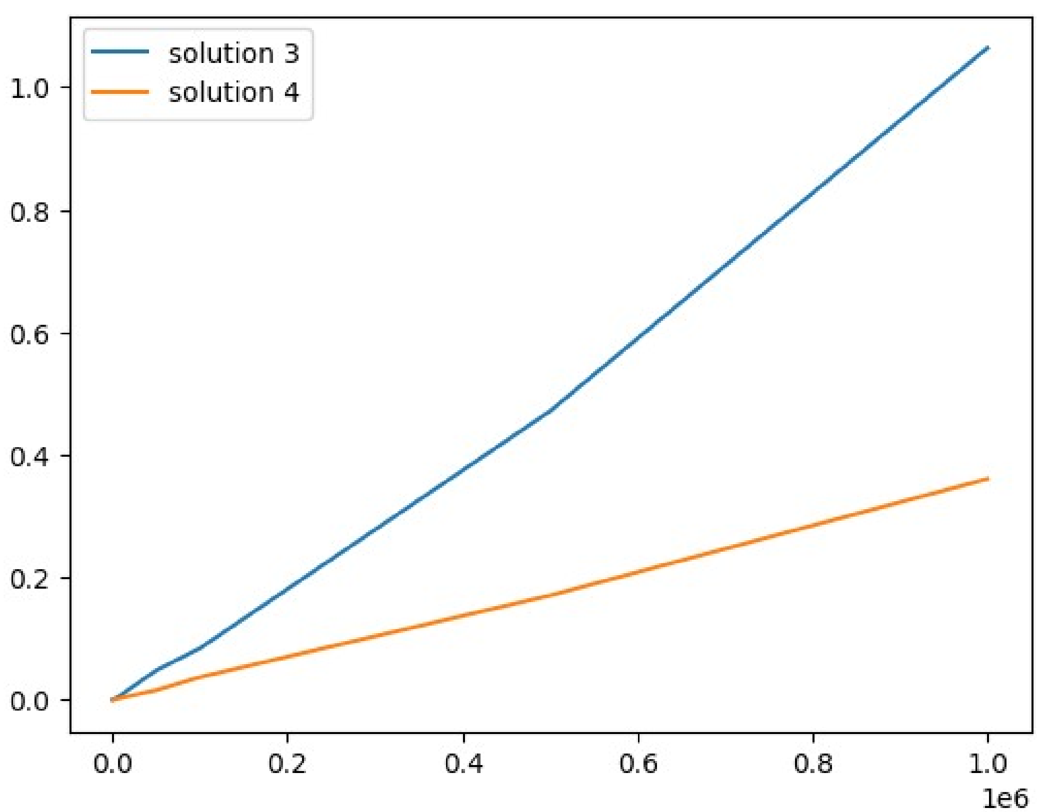
```
drawStuff()
```

Summary

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Solution 1	1e-05	1e-05	0.00012	0.00046	0.01204	0.05644	1.55748	5.79519	147.73879	595.15023	2380.60093	9522.40372
Solution 2	1e-05	1e-05	5e-05	0.00012	0.00099	0.00252	0.02401	0.06474	0.7863	2.06484	26.44584	72.89753
Solution 3	1e-05	1e-05	3e-05	5e-05	0.00029	0.0006	0.0034	0.00832	0.048	0.08409	0.471	1.06391
Solution 4	0.0	1e-05	2e-05	3e-05	0.00014	0.00029	0.00152	0.00307	0.01582	0.03693	0.17017	0.36042

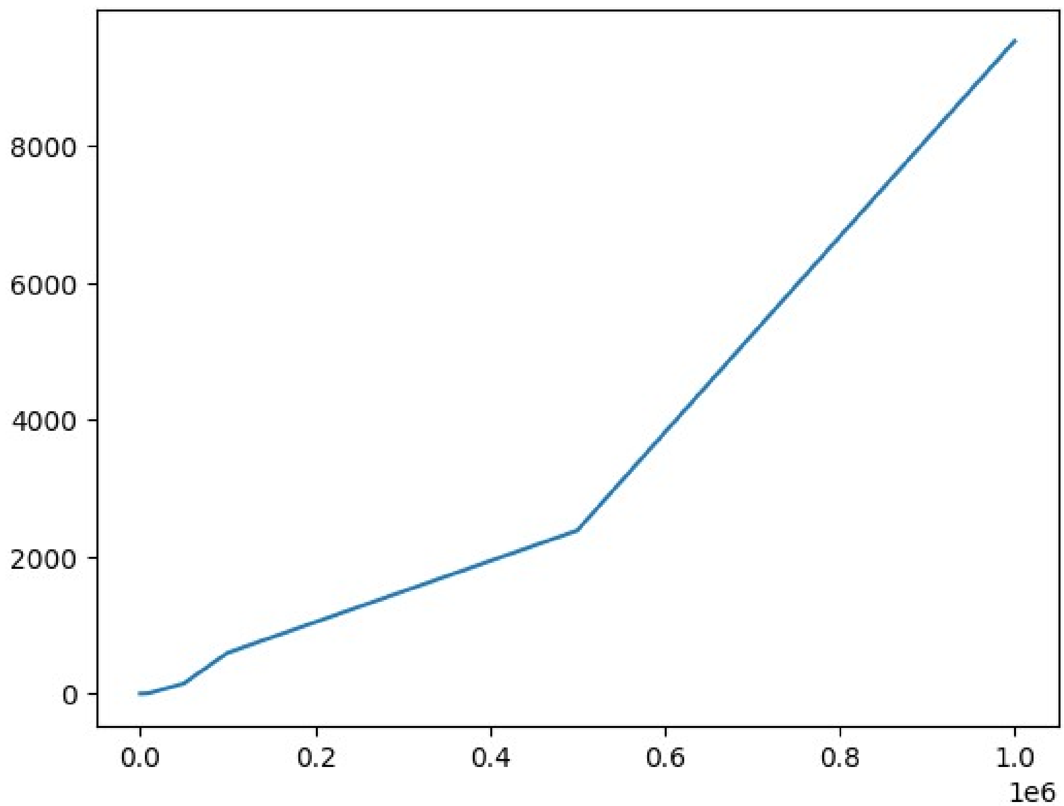



solution 2
solution 3
solution 4
70
60
50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
1e6


solution 3
solution 4
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1e6

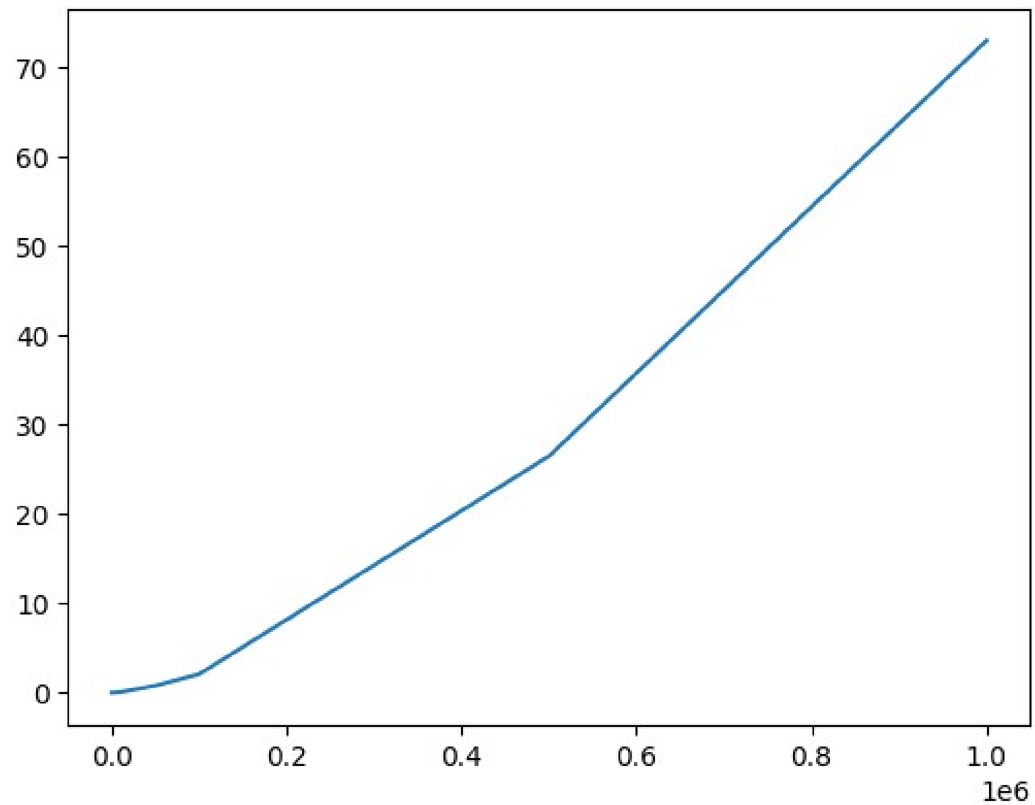
Solution 1

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Time	1e-05	1e-05	0.00012	0.00046	0.01204	0.05644	1.55748	5.79519	147.73879	595.15023	2380.60093	9522.40372



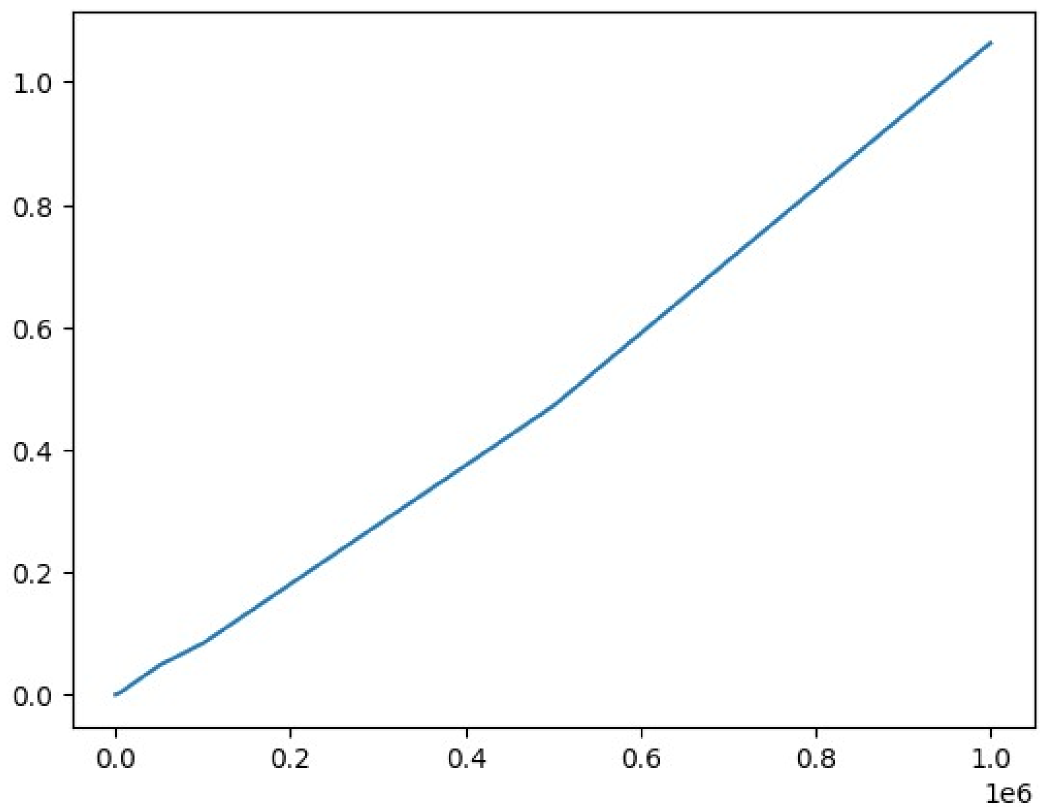
Solution 2

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Time	1e-05	1e-05	5e-05	0.00012	0.00099	0.00252	0.02401	0.06474	0.7863	2.06484	26.44584	72.89753



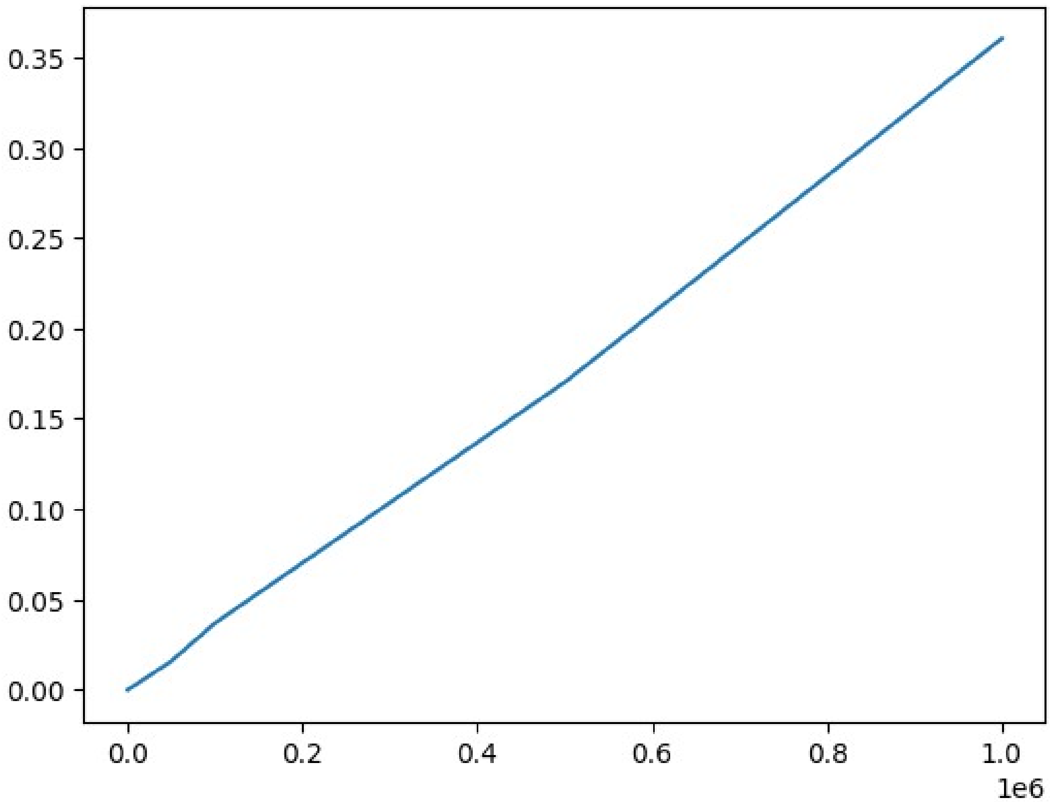
Solution 3

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Time	1e-05	1e-05	3e-05	5e-05	0.00029	0.0006	0.0034	0.00832	0.048	0.08409	0.471	1.06391



Solution 4

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Time	0.0	1e-05	2e-05	3e-05	0.00014	0.00029	0.00152	0.00307	0.01582	0.03693	0.17017	0.36042



In this part we convert boolean variables to int and will do someUnsufuleOperations() to make the proccess slower

Code with auto data, table and graph generate:

```
import math
import random
import time
import matplotlib.pyplot as plt
import numpy as np



def someUnsufuleOperations():
    p = 1
    p -= 1
    p += 1
    p = 0

def solution1(n):
    count = 0
    for i in range(2, n + 1):
        someUnsufuleOperations()
        isPrime = 1
        for j in range(2, i):
            if(i%j == 0):
                isPrime = 0
        if(isPrime == 1):
            count += 1
    return count

def solution2(n):
    count = 0
    for i in range(2, n + 1):
        someUnsufuleOperations()
        isPrime = 1
        for j in range(2, math.floor(math.sqrt(i))+1):
            if(i%j == 0):
                isPrime = 0
        if(isPrime == 1):
            count += 1
    return count

def solution3(n):
    # hasDivisor means divisors except 1 and the number itself
    hasDivisor = [0 for i in range(n+1)] # from 0 to n
    for i in range(2, n + 1):
        someUnsufuleOperations()
```

```
        for j in range(2 * i, n + 1, i):
            hasDivisor[j] = 1
    return n - hasDivisor.count(1) - 1 # -1 is for 1

def solution4(n):
    # hasDivisor means divisors except 1 and the number itself
    hasDivisor = [0 for i in range(n+1)] # from 0 to n
    for i in range(2, n + 1):
        someUnsufuleOperations()
        if(hasDivisor[i] == 0):
            for j in range(i * i, n + 1, i):
                hasDivisor[j] = 1
    return n - hasDivisor.count(1) - 1  #-1 is for 1

def drawGraph(x, y):
    plt.plot(x,y)
    plt.show()

def drawSummaryGraphs():
    for k in range(0, 3):
        for i in range(k, 4):
            plt.plot(inputs, times[i], label = "solution
{}".format(i+1))
        plt.legend()
        plt.show()

def drawTable(rows, columns, data, title):
    fig, ax = plt.subplots()
    ax.set_axis_off()
    rcolors = plt.cm.BuPu(np.full(len(rows), 0.1))
    ccolors = plt.cm.BuPu(np.full(len(columns), 0.1))
    table = ax.table(
        cellText = data,
        rowLabels = rows,
        colLabels = columns,
        rowColours = rcolors,
        colColours = ccolors,
        cellLoc ='center',
        loc ='upper left')

#     table.auto_set_font_size(False)
    table.set_fontsize(30)
    table.scale(2, 5)
    ax.set_title(title,
                 fontweight ="bold", fontdict={'fontsize': 30})

    plt.show()

def drawAllSingleTG():
    for i in range(4):
```

```
        drawTable(["Time"], inputs, [times[i]], 'Solution
{}'.format(i+1))
        drawGraph(inputs, times[i])

def calculateTimes():
    for n in inputs:
        start_time = time.time()
        solution1(n)
        times[0].append(time.time() - start_time)

        start_time = time.time()
        solution2(n)
        times[1].append(time.time() - start_time)

        start_time = time.time()
        solution3(n)
        times[2].append(time.time() - start_time)

        start_time = time.time()
        solution4(n)
        times[3].append(time.time() - start_time)

def drawStuff():
    drawTable(["Solution 1", "Solution 2", "Solution 3", "Solution
4"], inputs, times, 'Summary')
    drawSummaryGraphs()
    drawAllSingleTG()
    print("="*10 + " Times List " + "="*10)
    print(times)


inputs = [5, 10, 50, 100, 500, 10**3, 5 * 10**3, 10**4, 5 * 10**4,
10**5, 5 * 10**5, 10**6]
times = [[], [], [], []]
outputs = [[], [], [], []]
calculateTimes()
drawStuff()
```

Code with saved datas:

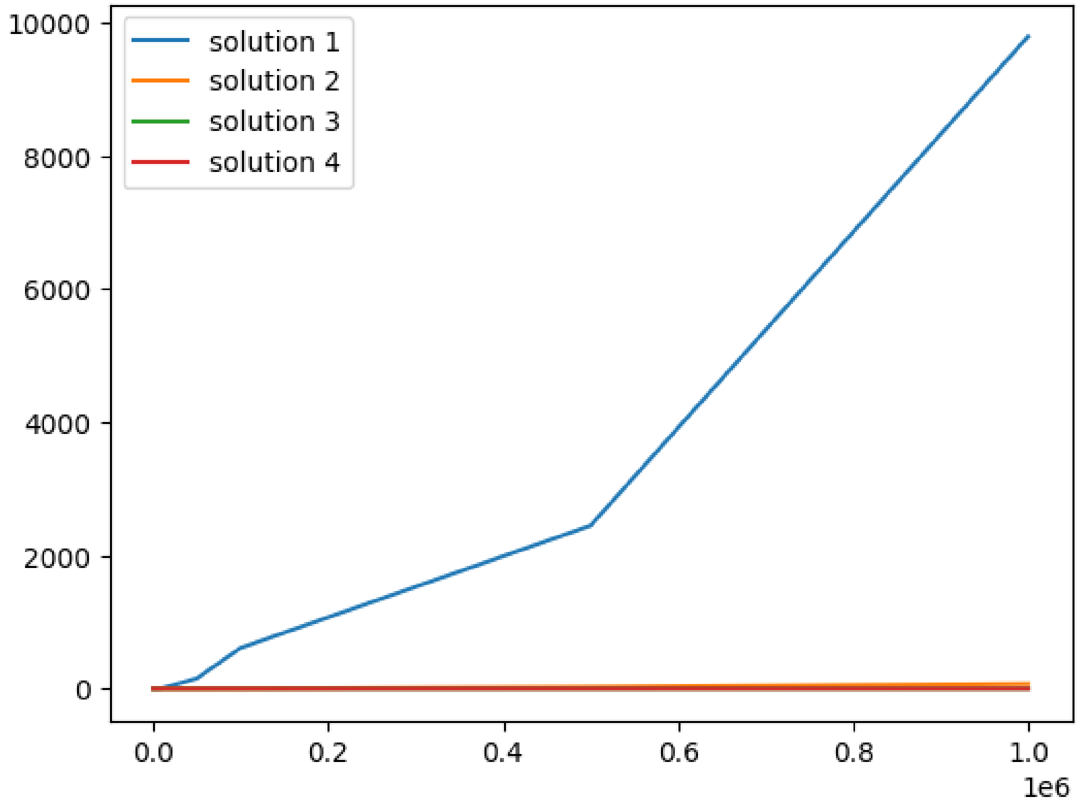
```
import matplotlib.pyplot as plt
import numpy as np

inputs = [5, 10, 50, 100, 500, 10**3, 5 * 10**3, 10**4, 5 * 10**4,
10**5, 5 * 10**5, 10**6]

times = [[1.71661376953125e-05, 1.7642974853515625e-05,
0.00021934509277734375, 0.0007827281951904297, 0.0176053047180117578,
0.05678439140319824, 1.64115309715271, 6.2975544929504395,
153.90738463401794, 612.1382052898407, 4*612.1382052898407,
```

```
16*612.1382052898407], [1.6927719116210938e-05, 1.7881393432617188e-
05, 0.00010609626770019531, 0.0002925395965576172,
0.00153350830078125, 0.002889394760131836, 0.031970977783203125,
0.06746912002563477, 0.6907429695129395, 1.9548075199127197,
25.54882311820984, 74.59105324745178], [1.5974044799804688e-05,
1.4543533325195312e-05, 6.103515625e-05, 0.0001220703125,
0.0006418228149414062, 0.0009315013885498047, 0.005592823028564453,
0.009818553924560547, 0.05376315116882324, 0.1100320816040039,
0.6264150142669678, 1.376746654510498], [1.0728836059570312e-05,
1.2636184692382812e-05, 4.601478576660156e-05, 8.606910705566406e-05,
0.0003707408905029297, 0.0010349750518798828, 0.004034519195556641,
0.00587916374206543, 0.02996516227722168, 0.059784889221191406,
0.30876660346984863, 0.6495921611785889]]

for i in range(len(times)):
    for j in range(len(times[i])):
        times[i][j] = float("{:.5f}".format(times[i][j]))

def drawGraph(x, y):
    plt.plot(x,y)
    plt.show()

def drawSummaryGraphs():
    for k in range(0, 3):
        for i in range(k, 4):
            plt.plot(inputs, times[i], label = "solution
{}".format(i+1))
        plt.legend()
        plt.show()

def drawTable(rows, columns, data, title):
    fig, ax = plt.subplots()
    ax.set_axis_off()
    rcolors = plt.cm.BuPu(np.full(len(rows), 0.1))
    ccolors = plt.cm.BuPu(np.full(len(columns), 0.1))
    table = ax.table(
        cellText = data,
        rowLabels = rows,
        colLabels = columns,
        rowColours = rcolors,
        colColours = ccolors,
        cellLoc ='center',
        loc ='upper left')

#     table.auto_set_font_size(False)
    table.set_fontsize(30)
    table.scale(2, 5)
    ax.set_title(title,
                 fontweight ="bold", fontdict={'fontsize': 30})
```

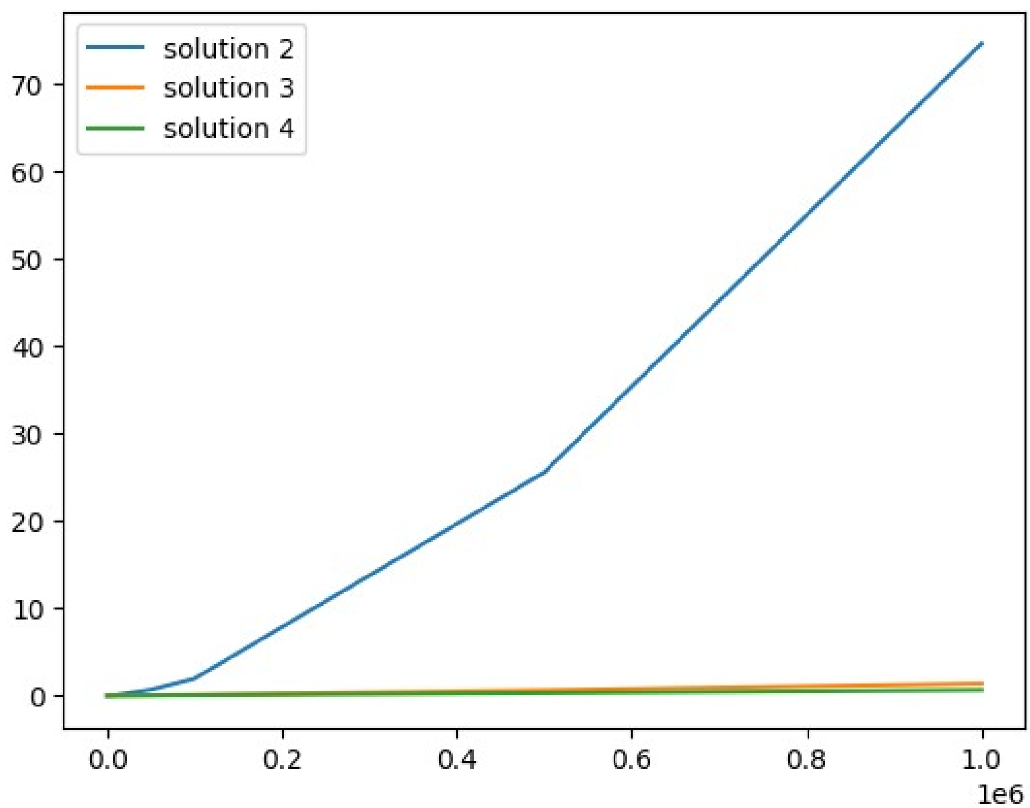
```
    plt.show()

    for i in range(len(data)):
        for j in range(len(data[i])):
            if(data[i][j] == "Too much"):
                data[i][j] = -1

def drawAllSingleGraphs():
    for i in range(4):
        drawTable(["Time"], inputs, [times[i]], 'Solution
{}'.format(i+1))
        drawGraph(inputs, times[i])

def drawStuff():
    drawTable(["Solution 1", "Solution 2", "Solution 3", "Solution
4"], inputs, times, 'Summary')
    drawSummaryGraphs()
    drawAllSingleGraphs()



drawStuff()
```

Summary

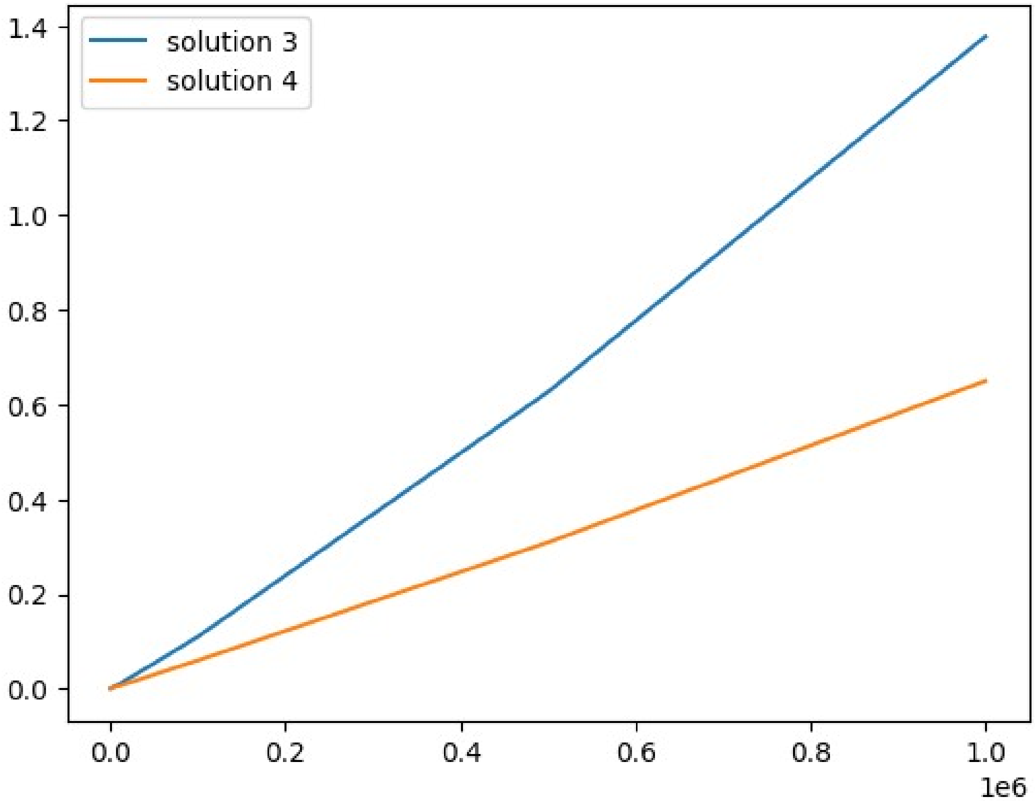
	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Solution 1	2e-05	2e-05	0.00022	0.00078	0.01761	0.05678	1.64115	6.29755	153.90738	612.13821	2448.55282	9794.21128
Solution 2	2e-05	2e-05	0.00011	0.00029	0.00153	0.00289	0.03197	0.06747	0.69074	1.95481	25.54882	74.59105
Solution 3	2e-05	1e-05	6e-05	0.00012	0.00064	0.00093	0.00559	0.00982	0.05376	0.11003	0.62642	1.37675
Solution 4	1e-05	1e-05	5e-05	9e-05	0.00037	0.00103	0.00403	0.00588	0.02997	0.05978	0.30877	0.64959



solution 1
solution 2
solution 3
solution 4
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
1e6

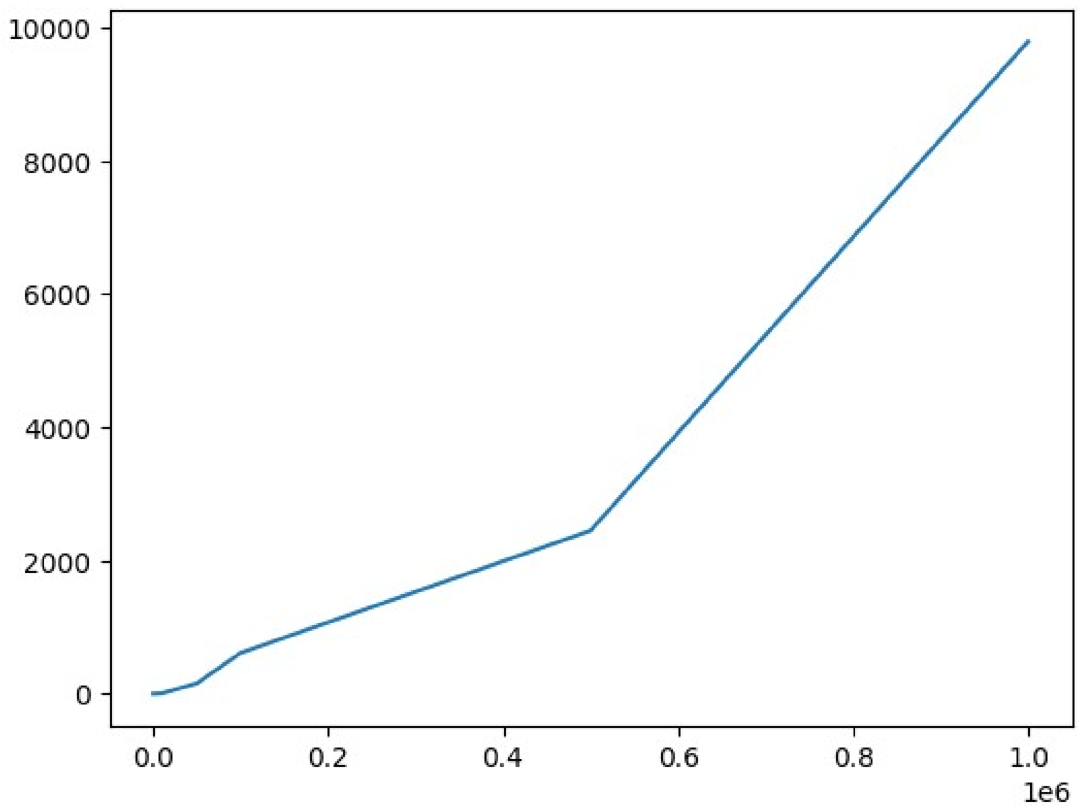


solution 2
solution 3
solution 4
70
60
50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
1e6



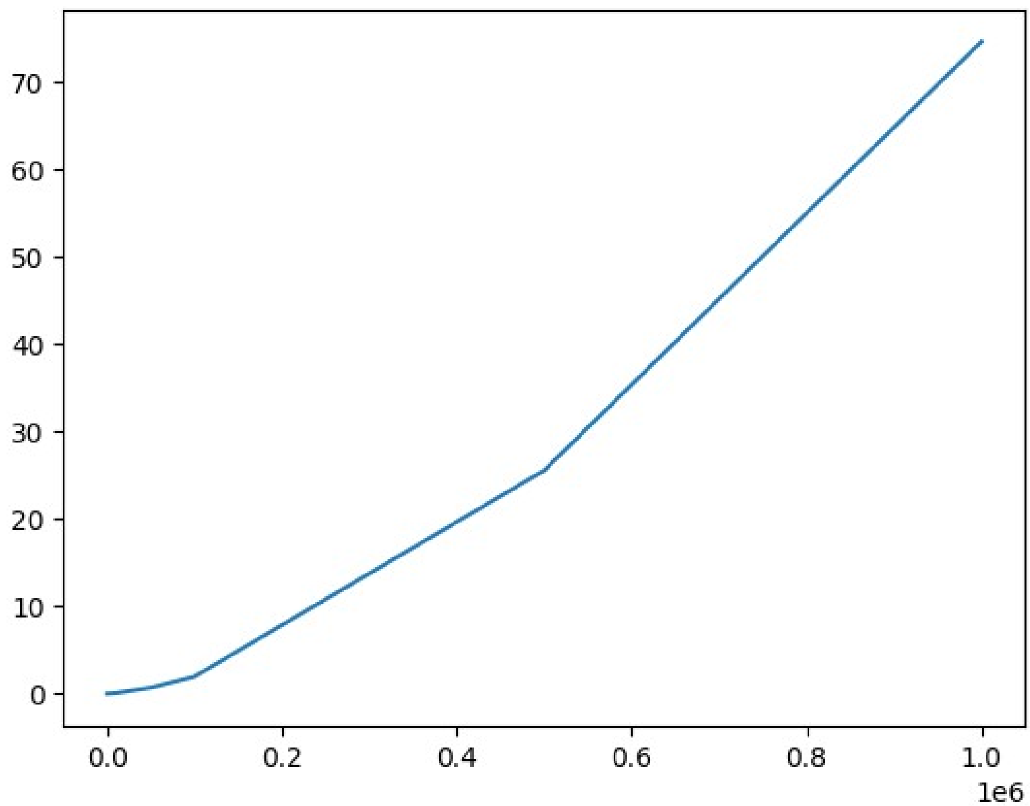
Solution 1

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Time	2e-05	2e-05	0.00022	0.00078	0.01761	0.05678	1.64115	6.29755	153.90738	612.13821	2448.55282	9794.21128



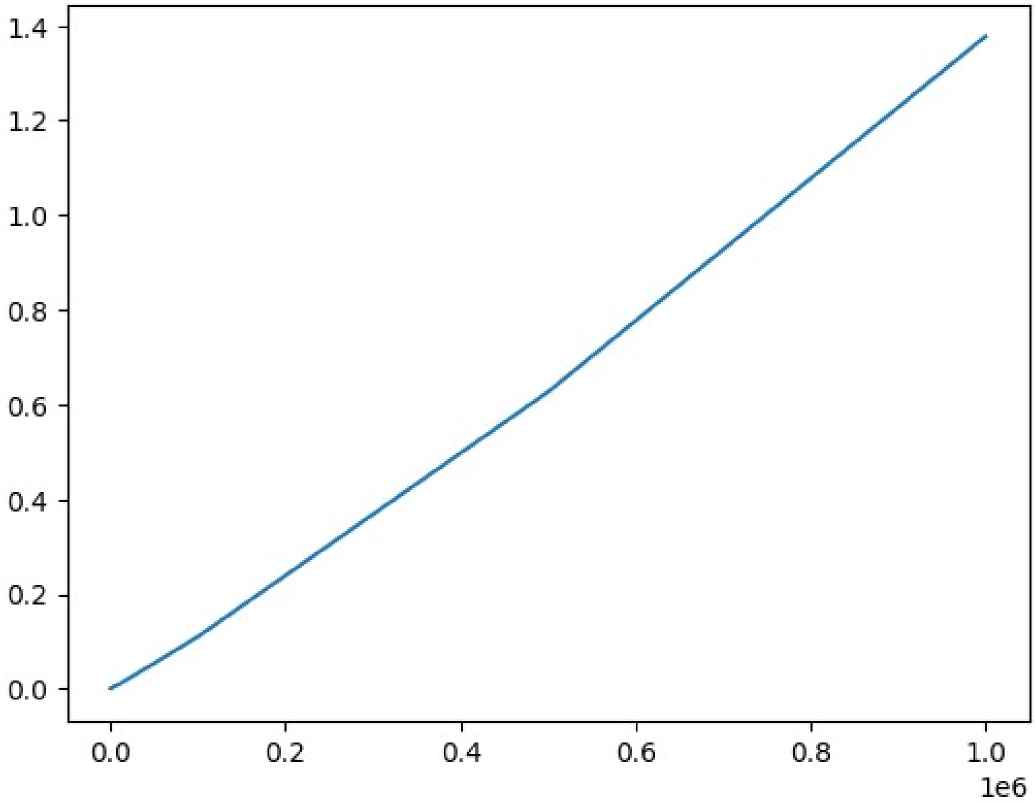
Solution 2

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Time	2e-05	2e-05	0.00011	0.00029	0.00153	0.00289	0.03197	0.06747	0.69074	1.95481	25.54882	74.59105



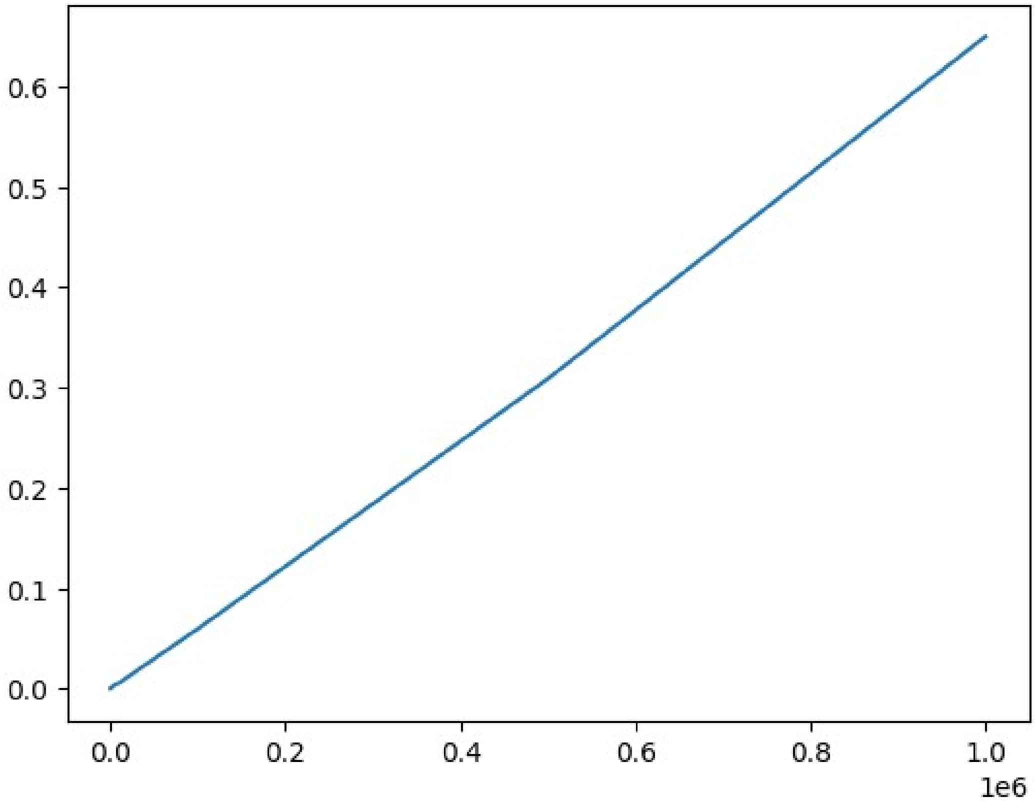
Solution 3

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Time	2e-05	1e-05	6e-05	0.00012	0.00064	0.00093	0.00559	0.00982	0.05376	0.11003	0.62642	1.37675



Solution 4

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Time	1e-05	1e-05	5e-05	9e-05	0.00037	0.00103	0.00403	0.00588	0.02997	0.05978	0.30877	0.64959



0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1e6

In this part we track times while running some heavy programs like Android Studio and A game (like graphical chess) :

Code with auto data, table and graph generate:

```
import math
import random
import time
import matplotlib.pyplot as plt
import numpy as np

def solution1(n):
    count = 0
    for i in range(2, n + 1):
        isPrime = True
        for j in range(2, i):
            if(i%j == 0):
                isPrime = False
        if(isPrime):
            count += 1
    return count

def solution2(n):
    count = 0
    for i in range(2, n + 1):
        isPrime = True
        for j in range(2, math.floor(math.sqrt(i))+1):
            if(i%j == 0):
                isPrime = False
        if(isPrime):
            count += 1
    return count

def solution3(n):
    # hasDivisor means divisors except 1 and the number itself
    hasDivisor = [False for i in range(n+1)] # from 0 to n
    for i in range(2, n + 1):
        for j in range(2 * i, n + 1, i):
            hasDivisor[j] = True
    return n - hasDivisor.count(True) - 1 # -1 is for 1

def solution4(n):
    # hasDivisor means divisors except 1 and the number itself
    hasDivisor = [False for i in range(n+1)] # from 0 to n
    for i in range(2, n + 1):
        if(not hasDivisor[i]):
            for j in range(i * i, n + 1, i):
                hasDivisor[j] = True
```

```
    return n - hasDivisor.count(True) - 1  #-1 is for 1

def drawGraph(x, y):
    plt.plot(x,y)
    plt.show()

def drawSummaryGraphs():
    for k in range(0, 3):
        for i in range(k, 4):
            plt.plot(inputs, times[i], label = "solution
{}".format(i+1))
        plt.legend()
        plt.show()

def drawTable(rows, columns, data, title):
    fig, ax = plt.subplots()
    ax.set_axis_off()
    rcolors = plt.cm.BuPu(np.full(len(rows), 0.1))
    ccolors = plt.cm.BuPu(np.full(len(columns), 0.1))
    table = ax.table(
        cellText = data,
        rowLabels = rows,
        colLabels = columns,
        rowColours = rcolors,
        colColours = ccolors,
        cellLoc ='center',
        loc ='upper left')

#     table.auto_set_font_size(False)
    table.set_fontsize(30)
    table.scale(2, 5)
    ax.set_title(title,
                 fontweight ="bold", fontdict={'fontsize': 30})

    plt.show()

def drawAllSingleTG():
    for i in range(4):
        drawTable(["Time"], inputs, [times[i]], 'Solution
{}'.format(i+1))
        drawGraph(inputs, times[i])

def drawStuff():
    drawTable(["Solution 1", "Solution 2", "Solution 3", "Solution
4"], inputs, times, 'Summary')
    drawSummaryGraphs()
    drawAllSingleTG()
    print("="*10 + " Times List " + "="*10)
    print(times)
```

```
def calculateTimes():
    for n in inputs:
        start_time = time.time()
        solution1(n)
        times[0].append(time.time() - start_time)

        start_time = time.time()
        solution2(n)
        times[1].append(time.time() - start_time)

        start_time = time.time()
        solution3(n)
        times[2].append(time.time() - start_time)

        start_time = time.time()
        solution4(n)
        times[3].append(time.time() - start_time)



inputs = [5, 10, 50, 100, 500, 10**3, 5 * 10**3, 10**4, 5 * 10**4,
10**5, 5 * 10**5, 10**6]
times = [[], [], [], []]
outputs = [[], [], [], []]
calculateTimes()
drawStuff()
```

Code with saved datas:

```
import matplotlib.pyplot as plt
import numpy as np

inputs = [5, 10, 50, 100, 500, 10**3, 5 * 10**3, 10**4, 5 * 10**4,
10**5, 5 * 10**5, 10**6]

times = [[8.106231689453125e-06, 8.344650268554688e-06,
0.00012636184692382812, 0.0004737377166748047, 0.012262821197509766,
0.0542604923248291, 1.4507133960723877, 6.291901350021362,
148.18718767166138, 591.2566955089569, 4.5*591.2566955089569,
4.5*4.5*591.2566955089569], [9.059906005859375e-06,
8.821487426757812e-06, 5.245208740234375e-05, 0.0001227855682373047,
0.0010223388671875, 0.002592325210571289, 0.02563929557800293,
0.07202458381652832, 0.6946070194244385, 1.9673988819122314,
25.00427770614624, 73.84481120109558], [7.867813110351562e-06,
7.3909759521484375e-06, 2.8848648071289062e-05, 6.079673767089844e-05,
0.0003972053527832031, 0.0008769035339355469, 0.005219936370849609,
0.012183427810668945, 0.05977797508239746, 0.12654423713684082,
0.7922322750091553, 1.867142915725708], [5.245208740234375e-06,
5.7220458984375e-06, 1.6450881958007812e-05, 2.8133392333984375e-05,
0.00014829635620117188, 0.0003020763397216797, 0.0015347003936767578,
```

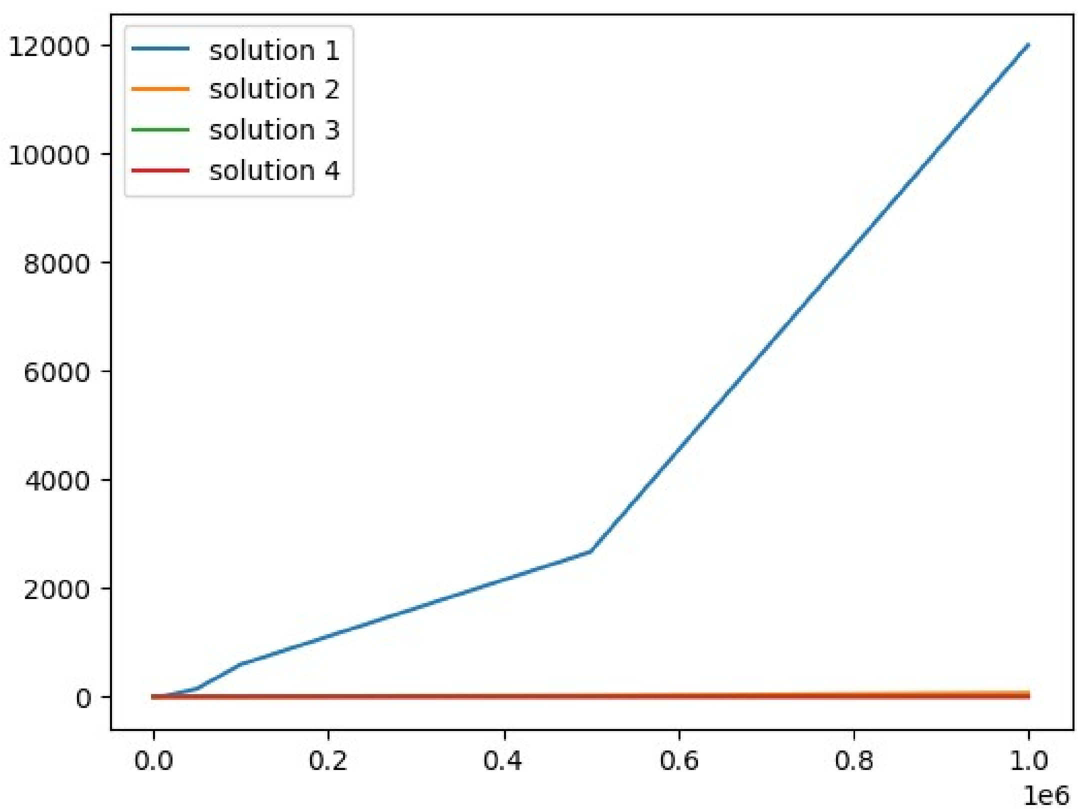
```
0.0031065940856933594, 0.015790224075317383, 0.03180360794067383,
0.17420434951782227, 0.36737847328186035]]
for i in range(len(times)):
    for j in range(len(times[i])):
        times[i][j] = float("{:.5f}".format(times[i][j]))

def drawGraph(x, y):
    plt.plot(x,y)
    plt.show()

def drawSummaryGraphs():
    for k in range(0, 3):
        for i in range(k, 4):
            plt.plot(inputs, times[i], label = "solution
{}".format(i+1))
        plt.legend()
        plt.show()

def drawTable(rows, columns, data, title):
    fig, ax = plt.subplots()
    ax.set_axis_off()
    rcolors = plt.cm.BuPu(np.full(len(rows), 0.1))
    ccolors = plt.cm.BuPu(np.full(len(columns), 0.1))
    table = ax.table(
        cellText = data,
        rowLabels = rows,
        colLabels = columns,
        rowColours = rcolors,
        colColours = ccolors,
        cellLoc ='center',
        loc ='upper left')

#     table.auto_set_font_size(False)
    table.set_fontsize(30)
    table.scale(2, 5)
    ax.set_title(title,
                 fontweight ="bold", fontdict={'fontsize': 30})

    plt.show()

def drawAllSingleGraphs():
    for i in range(4):
        drawTable(["Time"], inputs, [times[i]], 'Solution
{}'.format(i+1))
        drawGraph(inputs, times[i])

def drawStuff():
    drawTable(["Solution 1", "Solution 2", "Solution 3", "Solution
4"], inputs, times, 'Summary')
    drawSummaryGraphs()
```

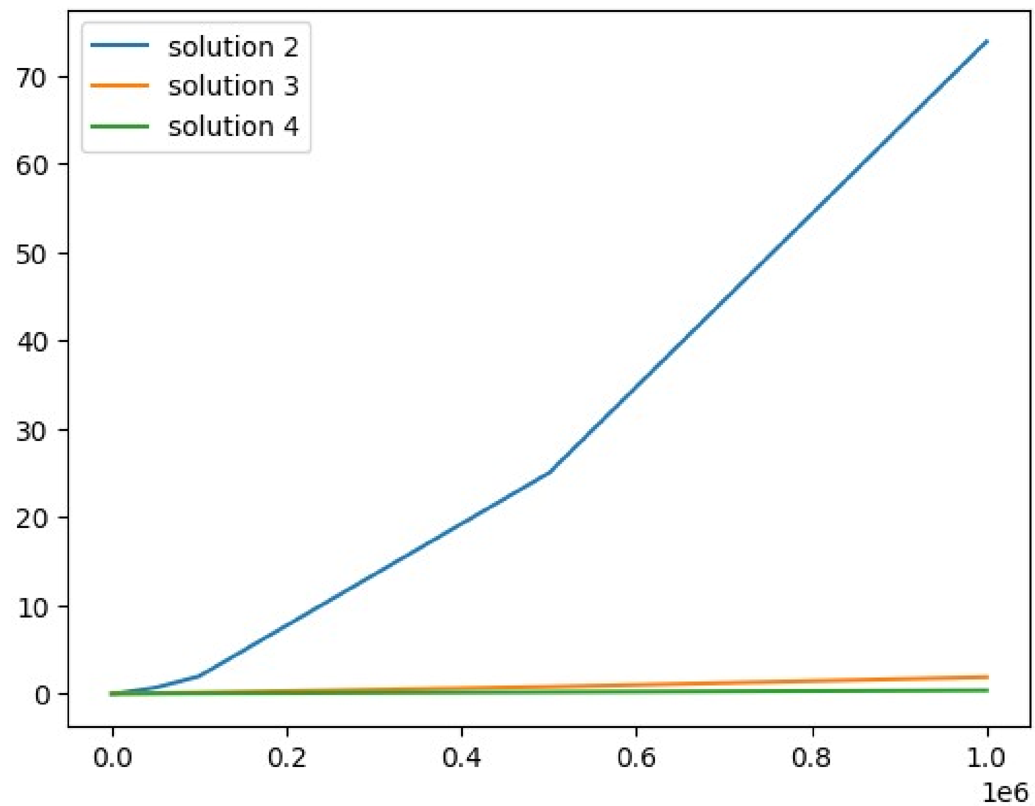
```
    drawAllSingleGraphs()
```

```
drawStuff()
```

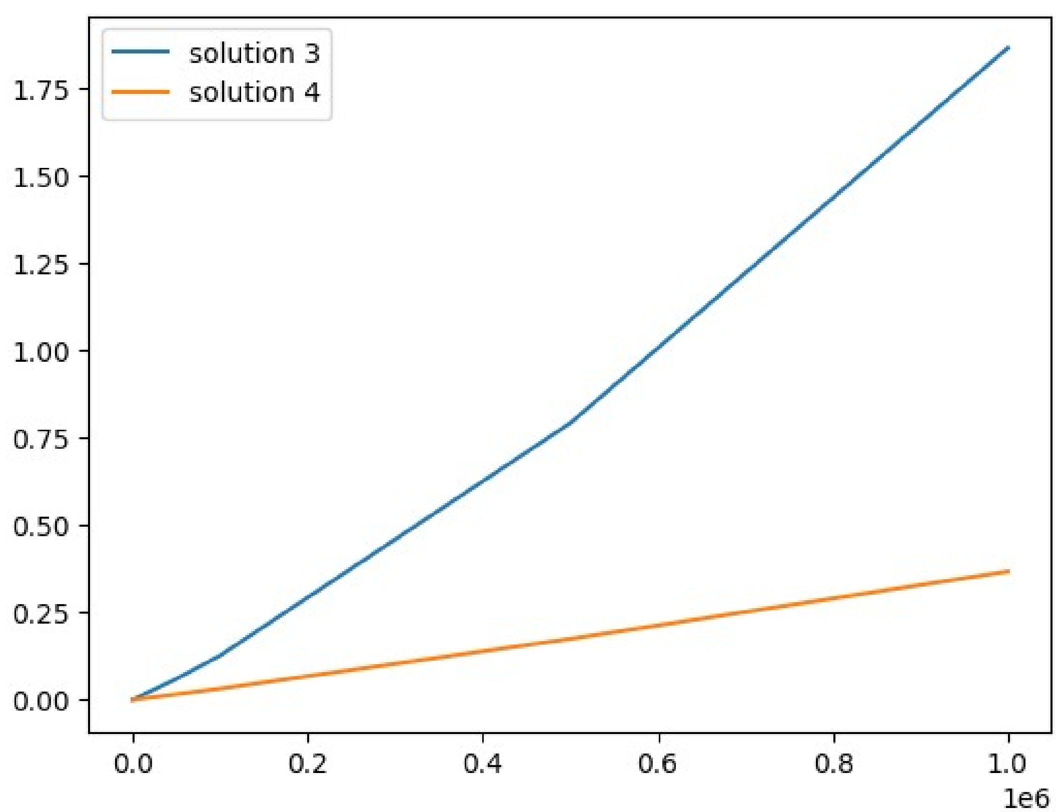
Summary

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Solution 1	1e-05	1e-05	0.00013	0.00047	0.01226	0.05426	1.45071	6.2919	148.18719	591.2567	2660.65513	11972.94808
Solution 2	1e-05	1e-05	5e-05	0.00012	0.00102	0.00259	0.02564	0.07202	0.69461	1.9674	25.00428	73.84481
Solution 3	1e-05	1e-05	3e-05	6e-05	0.0004	0.00088	0.00522	0.01218	0.05978	0.12654	0.79223	1.86714
Solution 4	1e-05	1e-05	2e-05	3e-05	0.00015	0.0003	0.00153	0.00311	0.01579	0.0318	0.1742	0.36738





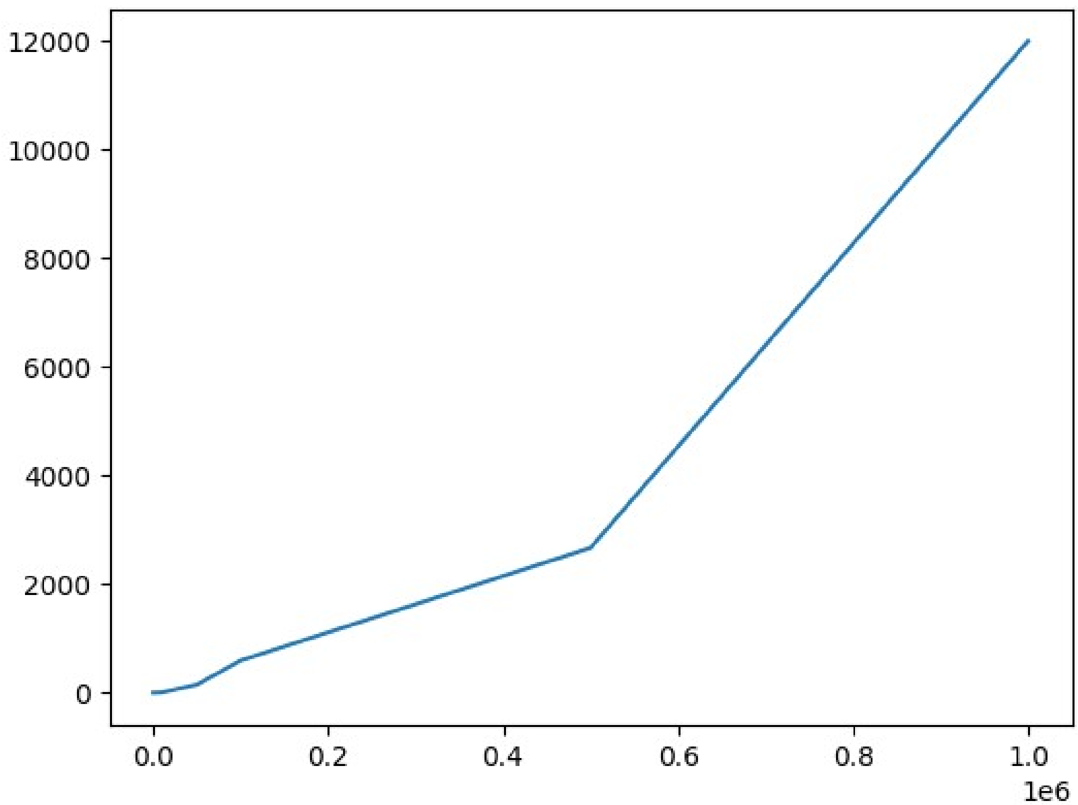
solution 2
solution 3
solution 4
70
60
50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
1e6



solution 3
solution 4
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0
1e6

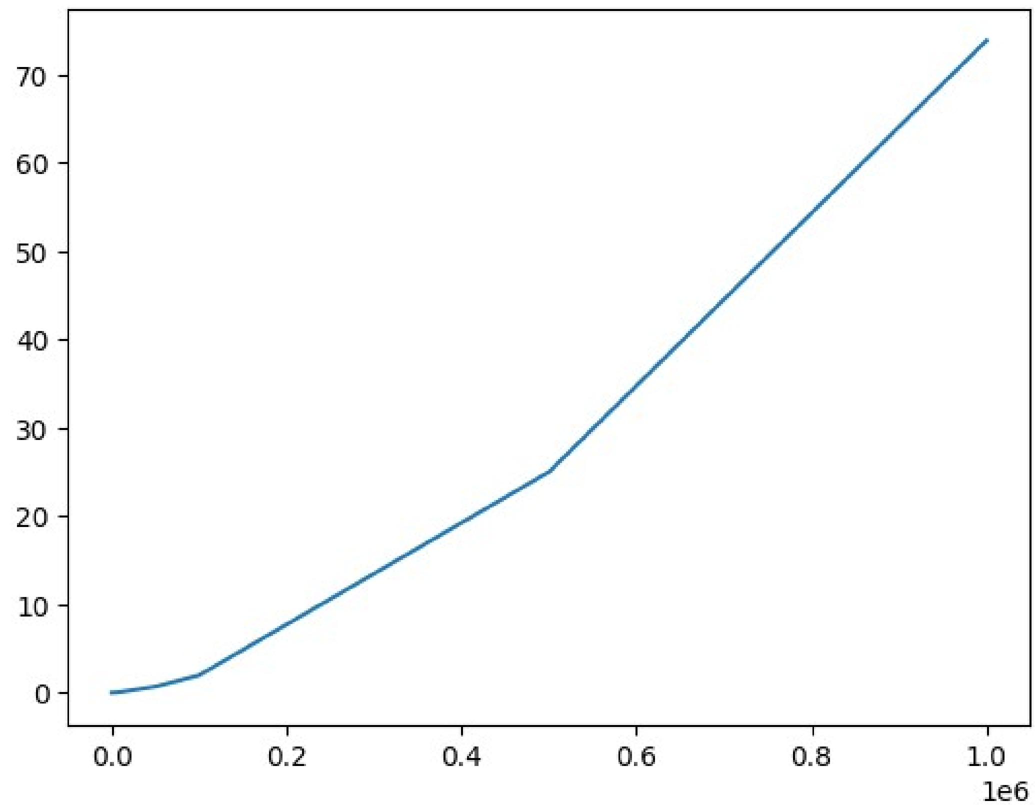
Solution 1

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Time	1e-05	1e-05	0.00013	0.00047	0.01226	0.05426	1.45071	6.2919	148.18719	591.2567	2660.65513	11972.94808



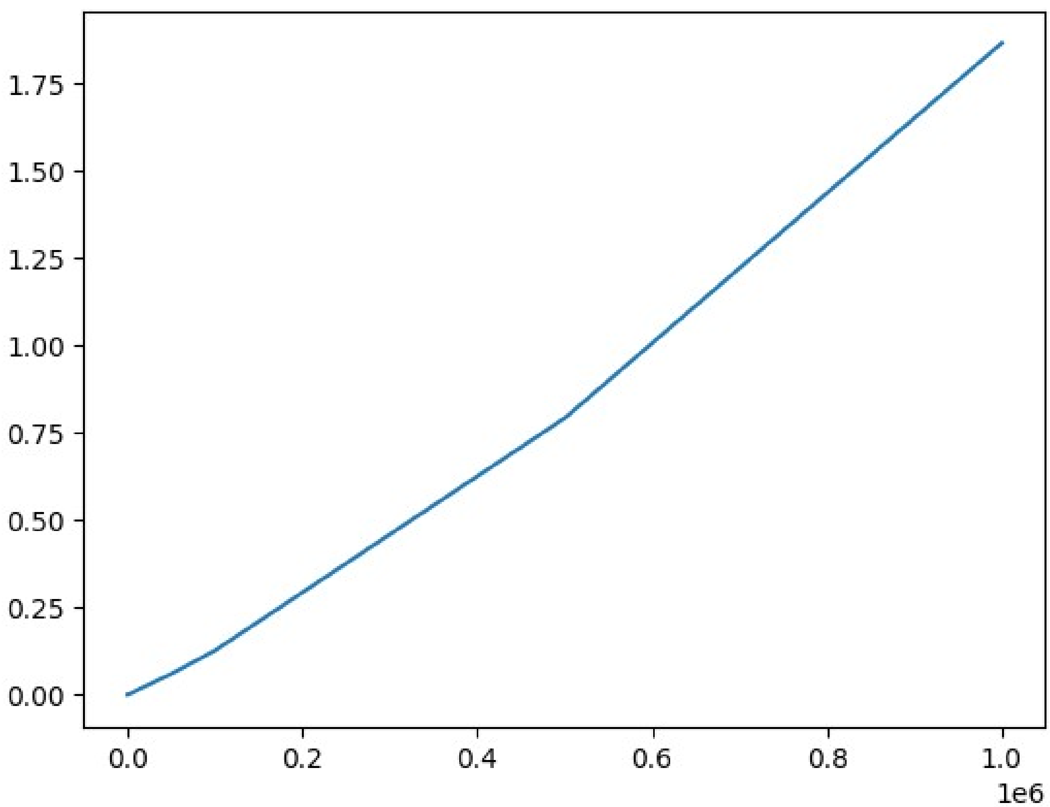
Solution 2

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Time	1e-05	1e-05	5e-05	0.00012	0.00102	0.00259	0.02564	0.07202	0.69461	1.9674	25.00428	73.84481



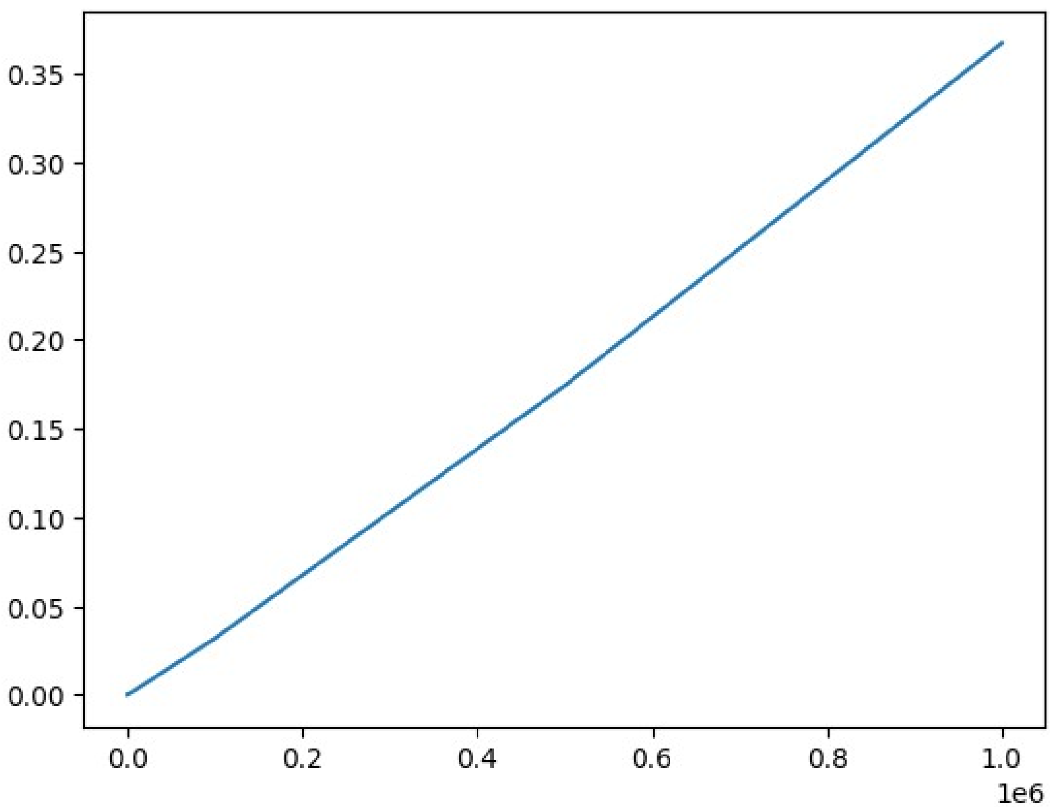
Solution 3

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Time	1e-05	1e-05	3e-05	6e-05	0.0004	0.00088	0.00522	0.01218	0.05978	0.12654	0.79223	1.86714



Solution 4

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
Time	1e-05	1e-05	2e-05	3e-05	0.00015	0.0003	0.00153	0.00311	0.01579	0.0318	0.1742	0.36738



A Comparison between A B C

```
import matplotlib.pyplot as plt
import numpy as np

inputs = [5, 10, 50, 100, 500, 10**3, 5 * 10**3, 10**4, 5 * 10**4,
10**5, 5 * 10**5, 10**6]

times = [[[8.58306884765625e-06, 8.344650268554688e-06,
0.0001227855682373047, 0.0004599094390869l406, 0.01203608512878418,
0.05643773078918457, 1.5574758052825928, 5.795194864273071,
147.73878645896912, 595.1502323150635, 595.1502323150635*4,
595.1502323150635*16], [1.2874603271484375e-05, 9.5367431640625e-06,
5.221366882324219e-05, 0.00012230873107910156, 0.0009887218475341797,
0.002524852752685547, 0.02401137351989746, 0.06473803520202637,
0.7862980365753174, 2.0648396015167236, 26.445839405059814,
72.89753484725952], [8.106231689453125e-06, 6.9141387939453125e-06,
2.5272369384765625e-05, 5.030632019042969e-05, 0.0002853870391845703,
0.0006022453308105469, 0.003395795822143554 7, 0.008321762084960938,
0.04799675941467285, 0.08408665657043457, 0.47100043296813965,
1.0639145374298096], [4.76837158203125e-06, 5.4836273193359375e-06,
1.5974044799804688e-05, 2.8371810913085938e-05,
0.00014400482177734375, 0.0002930164337158203, 0.0015158653259277344,
0.0030672550201416016, 0.015823841094970703, 0.03692960739135742,
0.17017102241516113, 0.360424280166626]]]
times.append([[1.71661376953125e-05, 1.7642974853515625e-05,
0.00021934509277343750, 0.0007827281951904297, 0.017605304718017578,
0.05678439140319824, 1.64115309715271, 6.2975544929504395,
153.90738463401794, 612.1382052898407, 4*612.1382052898407,
16*612.1382052898407], [1.6927719116210938e-05, 1.7881393432617188e-
05, 0.00010609626770019531, 0.0002925395965576172,
0.00153350830078125, 0.002889394760131836, 0.031970977783203125,
0.06746912002563477, 0.6907429695129395, 1.9548075199127197,
25.54882311820984, 74.59105324745178], [1.5974044799804688e-05,
1.4543533325195312e-05, 6.103515625e-05, 0.0001220703125,
0.0006418228149414062, 0.0009315013885498047, 0.005592823028564453,
0.009818553924560547, 0.05376315116882324, 0.1100320816040039,
0.6264150142669678, 1.376746654510498], [1.0728836059570312e-05,
1.2636184692382812e-05, 4.601478576660156e-05, 8.606910705566406e-05,
0.0003707408905029297, 0.0010349750518798828, 0.004034519195556641,
0.00587916374206543, 0.02996516227722168, 0.059784889221191406,
0.30876660346984863, 0.6495921611785889]])
times.append([[8.106231689453125e-06, 8.344650268554688e-06,
0.00012636184692382812, 0.0004737377166748047, 0.012262821197509766,
0.0542604923248291, 1.4507133960723877, 6.291901350021362,
148.18718767166138, 591.2566955089569, 4.5*591.2566955089569,
4.5*4.5*591.2566955089569], [9.059906005859375e-06,
8.821487426757812e-06, 5.245208740234375e-05, 0.0001227855682373047,
0.0010223388671875, 0.002592325210571289, 0.02563929557800293,
```

```
0.07202458381652832, 0.6946070194244385, 1.9673988819122314,
25.00427770614624, 73.84481120109558], [7.867813110351562e-06,
7.3909759521484375e-06, 2.8848648071289062e-05, 6.079673767089844e-05,
0.0003972053527832031, 0.0008769035339355469, 0.005219936370849609,
0.012183427810668945, 0.05977797508239746, 0.12654423713684082,
0.7922322750091553, 1.867142915725708], [5.245208740234375e-06,
5.7220458984375e-06, 1.6450881958007812e-05, 2.8133392333984375e-05,
0.00014829635620117188, 0.00030207633972167969, 0.0015347003936767578,
0.0031065940856933594, 0.015790224075317383, 0.03180360794067383,
0.17420434951782227, 0.36737847328186035]])


for i in range(len(times)):
    for j in range(len(times[i])):
        for k in range(len(times[i][j])):
            times[i][j][k] = float("{:.5f}".format(times[i][j][k]))

def drawGraph(x, y):
    plt.plot(x,y)
    plt.show()

def drawSummaryGraphs():
    parts = ['A', 'B', 'C']
    for j in range(4):
        plt.title("Solution {}".format(j+1))
        for i in range(3):
            plt.plot(inputs, times[i][j], label =
"{}".format(parts[i]))
        plt.legend()
        plt.show()

def drawTable(rows, columns, data, title):
    fig, ax = plt.subplots()
    ax.set_axis_off()
    rcolors = plt.cm.BuPu(np.full(len(rows), 0.1))
    ccolors = plt.cm.BuPu(np.full(len(columns), 0.1))
    table = ax.table(
        cellText = data,
        rowLabels = rows,
        colLabels = columns,
        rowColours = rcolors,
        colColours = ccolors,
        cellLoc ='center',
        loc ='upper left')

#     table.auto_set_font_size(False)
    table.set_fontsize(30)
    table.scale(2, 5)
    ax.set_title(title,
                 fontweight ="bold", fontdict={'fontsize': 30})
```

```
    plt.show()

def drawStuff():
    for i in range(4):
        temp = []
        for j in range(3):
            temp.append(times[j][i])
        drawTable([" A ", " B ", " C "], inputs, temp, 'Solution 
{}'.format(i+1))
    drawSummaryGraphs()



drawStuff()
```

Solution 1

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
A	1e-05	1e-05	0.00012	0.00046	0.01204	0.05644	1.55748	5.79519	147.73879	595.15023	2380.60093	9522.40372
B	2e-05	2e-05	0.00022	0.00078	0.01761	0.05678	1.64115	6.29755	153.90738	612.13821	2448.55282	9794.21128
C	1e-05	1e-05	0.00013	0.00047	0.01226	0.05426	1.45071	6.2919	148.18719	591.2567	2660.65513	11972.94808

Solution 2

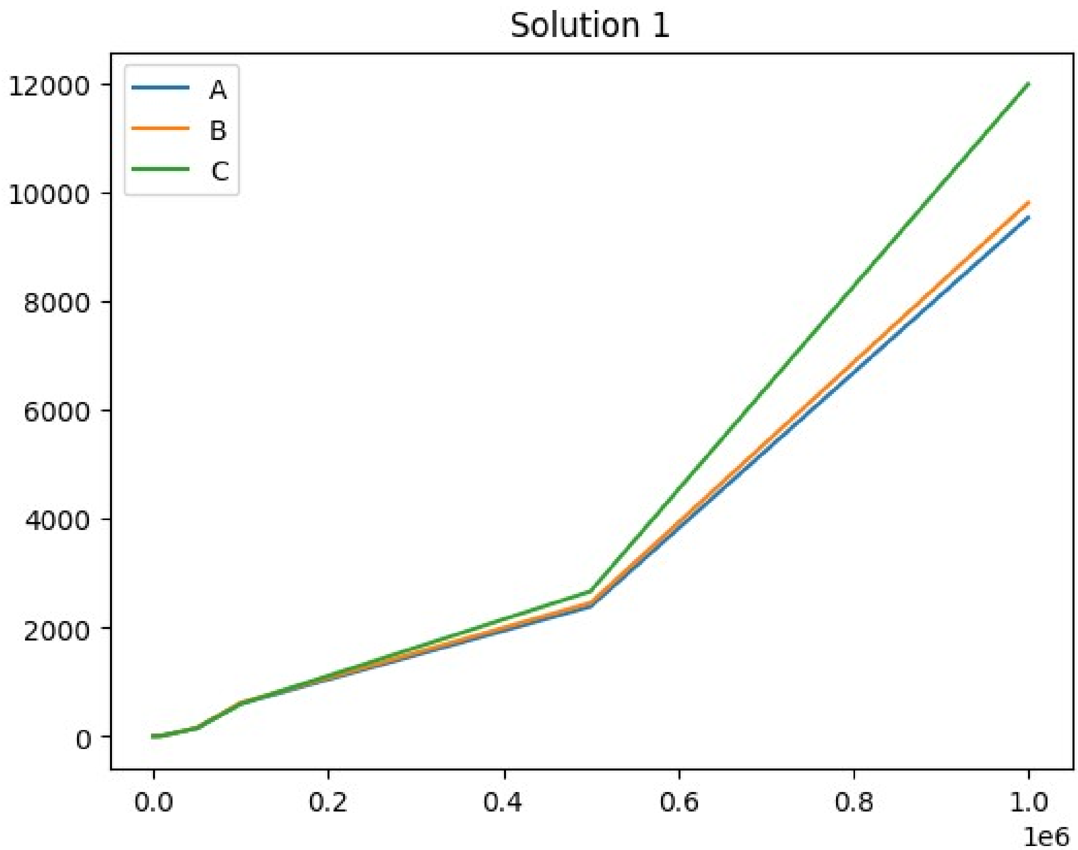
	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
A	1e-05	1e-05	5e-05	0.00012	0.00099	0.00252	0.02401	0.06474	0.7863	2.06484	26.44584	72.89753
B	2e-05	2e-05	0.00011	0.00029	0.00153	0.00289	0.03197	0.06747	0.69074	1.95481	25.54882	74.59105
C	1e-05	1e-05	5e-05	0.00012	0.00102	0.00259	0.02564	0.07202	0.69461	1.9674	25.00428	73.84481

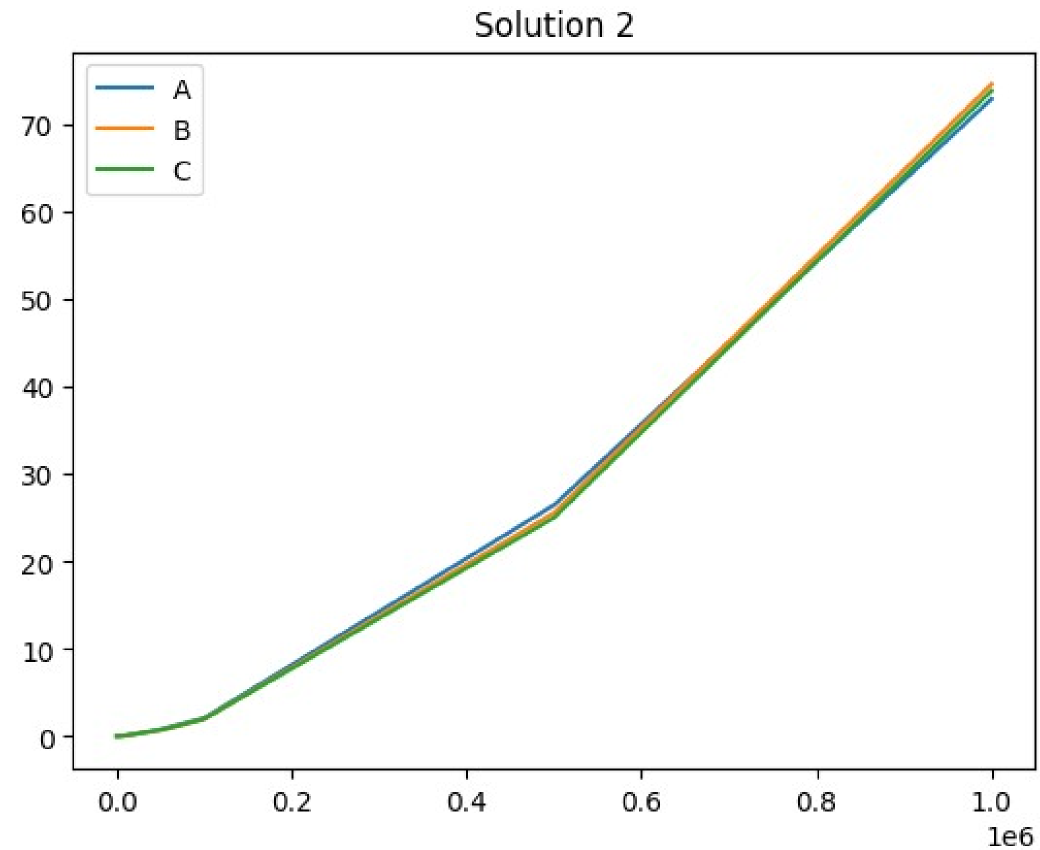
Solution 3

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
A	1e-05	1e-05	3e-05	5e-05	0.00029	0.0006	0.0034	0.00832	0.048	0.08409	0.471	1.06391
B	2e-05	1e-05	6e-05	0.00012	0.00064	0.00093	0.00559	0.00982	0.05376	0.11003	0.62642	1.37675
C	1e-05	1e-05	3e-05	6e-05	0.0004	0.00088	0.00522	0.01218	0.05978	0.12654	0.79223	1.86714

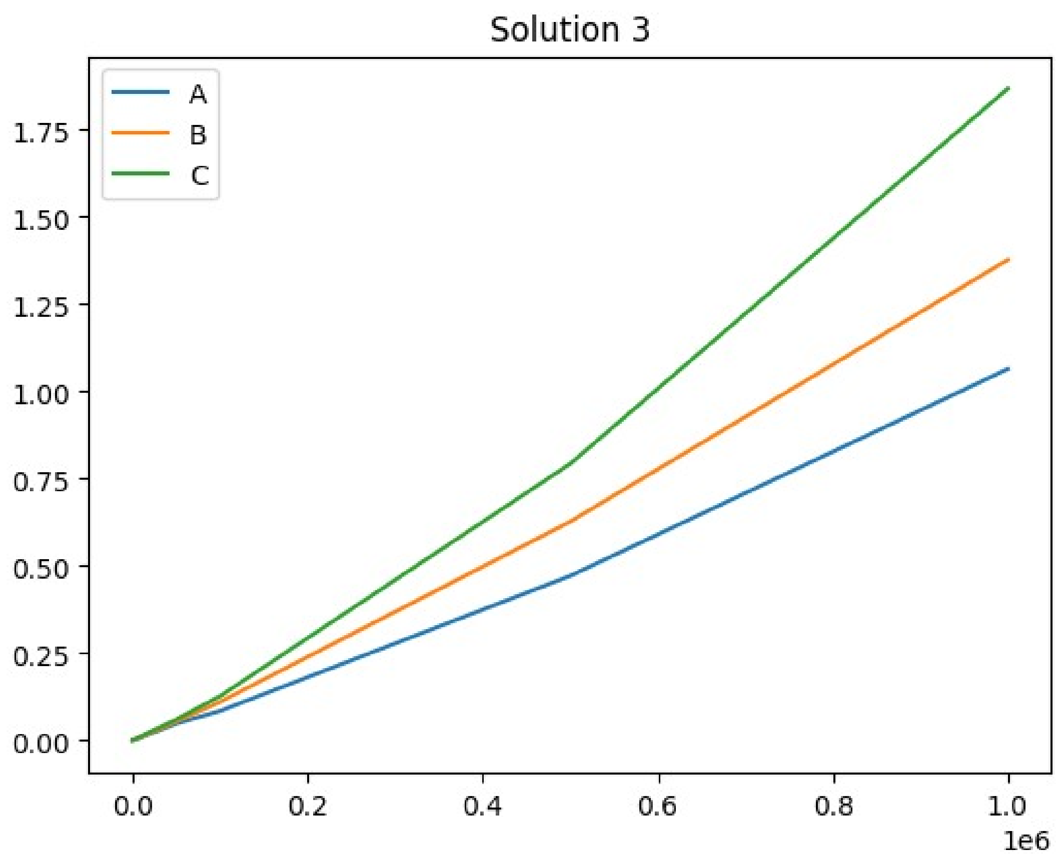
Solution 4

	5	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000
A	0.0	1e-05	2e-05	3e-05	0.00014	0.00029	0.00152	0.00307	0.01582	0.03693	0.17017	0.36042
B	1e-05	1e-05	5e-05	9e-05	0.00037	0.00103	0.00403	0.00588	0.02997	0.05978	0.30877	0.64959
C	1e-05	1e-05	2e-05	3e-05	0.00015	0.0003	0.00153	0.00311	0.01579	0.0318	0.1742	0.36738

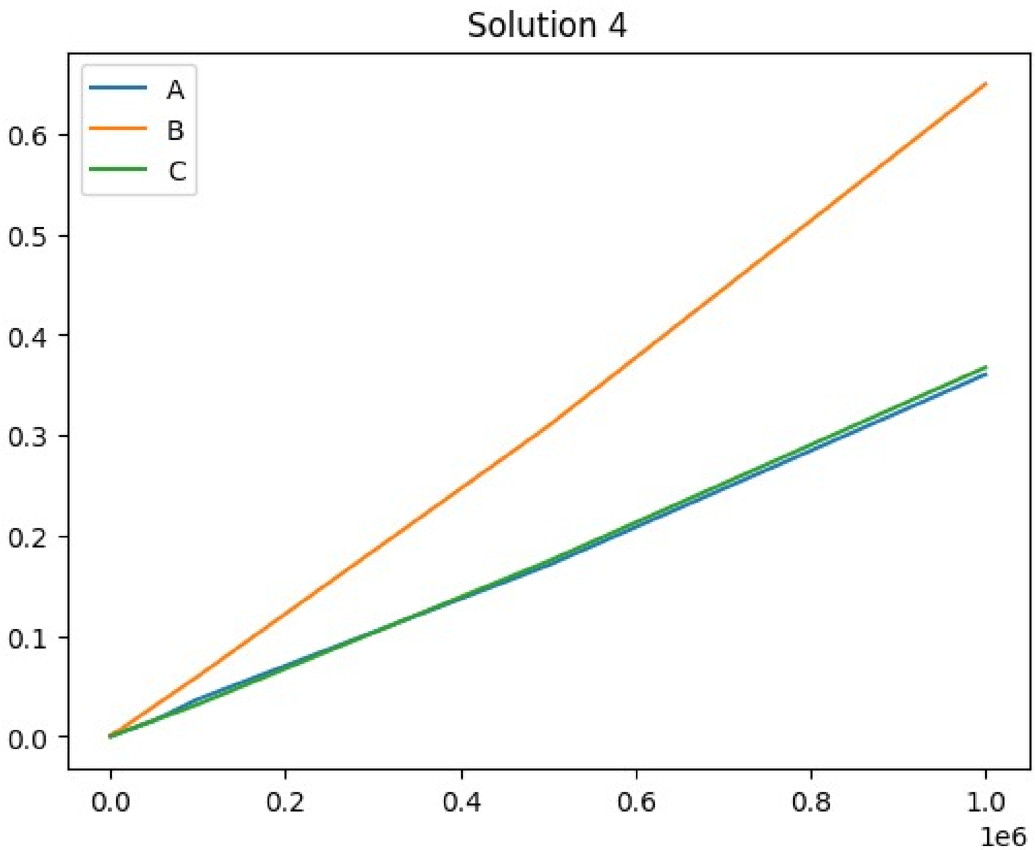

Solution 1
A
B
C
12000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
1e6



Solution 2
A
B
C
0
10
20
30
40
50
60
70
0.0
0.2
0.4
0.6
0.8
1.0
1e6



Solution 3
A
B
C
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0
1e6


Solution 4
A
B
C
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
0.2
0.4
0.6
0.8
1.0
1e6