Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота № 2

з дисципліни «Теорія алгоритмів»

на тему:

**"Алгоритм сортування включенням"**

Виконав:

студент гр. ІС-02

Павлущенко Ольга

Викладач:

Новікова П. А.

Київ – 2021

**Завдання**

1) Реалізувати алгоритми сортування методами бульбашки ( функцію bubble\_sort, improve\_bubble\_sort) та включення (функцію insertion\_sort). При чому під час роботи алгоритму повинні підраховуватись порівняння елементів, які виконує кожен з методів.

2) Провести тестування алгоритмів на різних наборах даних. Для тестування використовується функція generate\_data. Тестування проводиться для розмірів даних [10, 100, 1000, 10000] та трьох типів вхідних даних (випадково згенеровані вхідні дані, найкращі та найгірші вхідні дані). Порівняти отримані результати тестування.

3) Знайти спосіб покращити поведінку метода бульбашки, щоб він використовував меншу кількість операцій. Реалізувати знайдений спосіб та дійти висновку на скільки це вплинуло на якість роботи алгоритму

**Програмний код**

**Lab\_2.py**

import random

import copy

import numpy as np

from graphics import plot\_data

def main():

sizes = [10, 100, 1000]

types = ["random", "best", "worst"]

data\_plot = {'random': {'bubble':{}, 'insertion':{}, 'bubble\_impr':{}},

'best': {'bubble':{}, 'insertion':{}, 'bubble\_impr':{}},

'worst': {'bubble':{}, 'insertion':{}, 'bubble\_impr':{}}}

for n in sizes:

print ("\nDATA SIZE: ", n)

for gen\_type in types:

print ("\n\tDATA TYPE:", gen\_type)

data = generate\_data(n, gen\_type)

data\_bubble = np.copy(data)

bubble\_op\_count = buble\_sort(data\_bubble)

print ("\tBubble sort operation count:", int(bubble\_op\_count))

data\_plot[gen\_type]['bubble'][n] = bubble\_op\_count

data\_bubble\_impr = np.copy(data)

bubble\_impr\_op\_count = improved\_buble\_sort(data\_bubble\_impr)

print ("\tImproved bubble sort operation count:", int(bubble\_impr\_op\_count))

data\_plot[gen\_type]['bubble\_impr'][n] = bubble\_impr\_op\_count

data\_insertion = np.copy(data)

insertion\_op\_count = insertion\_sort(data\_insertion)

print ("\tInsertion sort operation count:", int(insertion\_op\_count))

data\_plot[gen\_type]['insertion'][n] = insertion\_op\_count

plot\_data(data\_plot, logarithmic=True, oneplot=False)

def buble\_sort(numbers):

n = len(numbers) #кількість елементів масиву

i = 0 #лічильник кількості проходів по масиву

count = 0 #лічильник для кількості порівнянь

for i in range(n):

for j in range( n-1): #проходження по елементам

count += 1

if numbers[j]>numbers[j+1]:

numbers[j],numbers[j+1]=numbers[j+1],numbers[j]

return count

ef insertion\_sort(num):

n = len(num)

count = 0 #лічильник для кількості порівнянь

for i in range(1,n):

temp = num[i] #змінна поточного елемента

j = i-1 #лічильник попередньго елемента

while(j>=0 and num[j]>temp):

num[j+1] = num[j]

j -= 1

if j>=0:

count+=1

num[j+1] = temp

count+=1

return count

def improved\_buble\_sort(numbers):

n = len(numbers)

count = 0 #лічильник кількості порівнянь

for i in range(n): #лічильник кількості проходів по масиву

swap = False

for j in range(n-i-1): #проходження по елементам

count += 1

if numbers[j]>numbers[j+1]:

numbers[j],numbers[j+1]=numbers[j+1],numbers[j]

swap = True

if swap == False:

break

return count

def generate\_data(n, gen\_type):

if gen\_type == "best": #генерація списку

A = [i+1 for i in range(n)]

#print(A)

return A

elif gen\_type == "worst": #генерація списку в зворотній послідовності

A = [i+1 for i in reversed(range(n))]

#print(A)

return A

else: #генерація рандомних значень

A = [i+1 for i in range(n)]

random.shuffle(A)

#print(A)

return A

main()

**graphics.py**

import collections

from pylab import figure, show

from math import log

def plot\_data(data, logarithmic=False, oneplot=False):

fig = figure(3)

num = len(data)

colors = ['r','g','y']

markers = ['s','o','v']

lines = ['-','---',':']

if oneplot is True:

ax = fig.add\_subplot(110)

ax.grid(True)

i = -1

line\_titles = []

x\_max = y\_max = 0

for label, value in data.items():

i += 1

j = -1

for sort\_type, points in value.items():

j += 1

od\_points = collections.OrderedDict(sorted(points.items()))

if logarithmic:

xs = [(x>0 and log(x,10) or 0) for x in od\_points.keys()]

ys = [(y>0 and log(y,10) or 0) for y in od\_points.values()]

else:

xs = od\_points.keys()

ys = od\_points.values()

xs.insert(0, 0)

x\_max = max(x\_max, max(xs))

ys.insert(0, 0)

y\_max = max(y\_max, max(ys))

ax.plot(xs, ys, colors[j%num]+markers[j%num]+lines[i%num], label=sort\_type )

line\_titles.append(sort\_type+' '+label)

ax.set\_xlim( (0, x\_max\*1.1) )

ax.set\_ylim( (0, y\_max\*1.1) )

ax.legend(line\_titles, loc=4)

else:

i = 0

for label, value in data.items():

i += 1

ax = fig.add\_subplot(num,1,i)

ax.grid(True)

ax.set\_title(label)

j = -1

x\_max = y\_max = 0

for sort\_type, points in value.items():

j += 1

od\_points = collections.OrderedDict(sorted(points.items()))

if logarithmic:

xs = [log(x,10) for x in od\_points.keys()]

ys = [log(y,10) for y in od\_points.values()]

else:

xs = od\_points.keys()

ys = od\_points.values()

xs.insert(0, 0)

x\_max = max(x\_max, max(xs))

ys.insert(0,0)

y\_max = max(y\_max, max(ys))

ax.plot(xs, ys, colors[j%num]+markers[j%num]+'-', label=sort\_type )

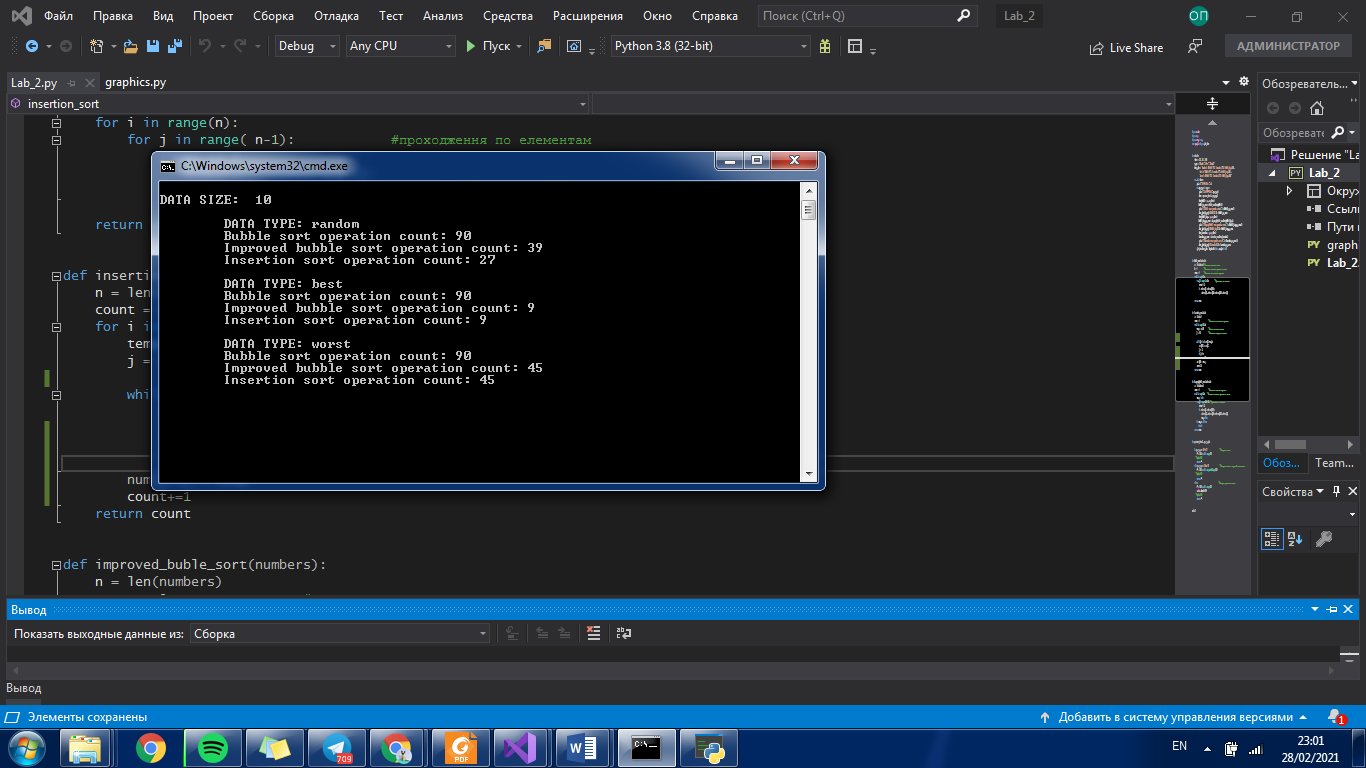
ax.set\_xlim( (0, x\_max\*1.2) )

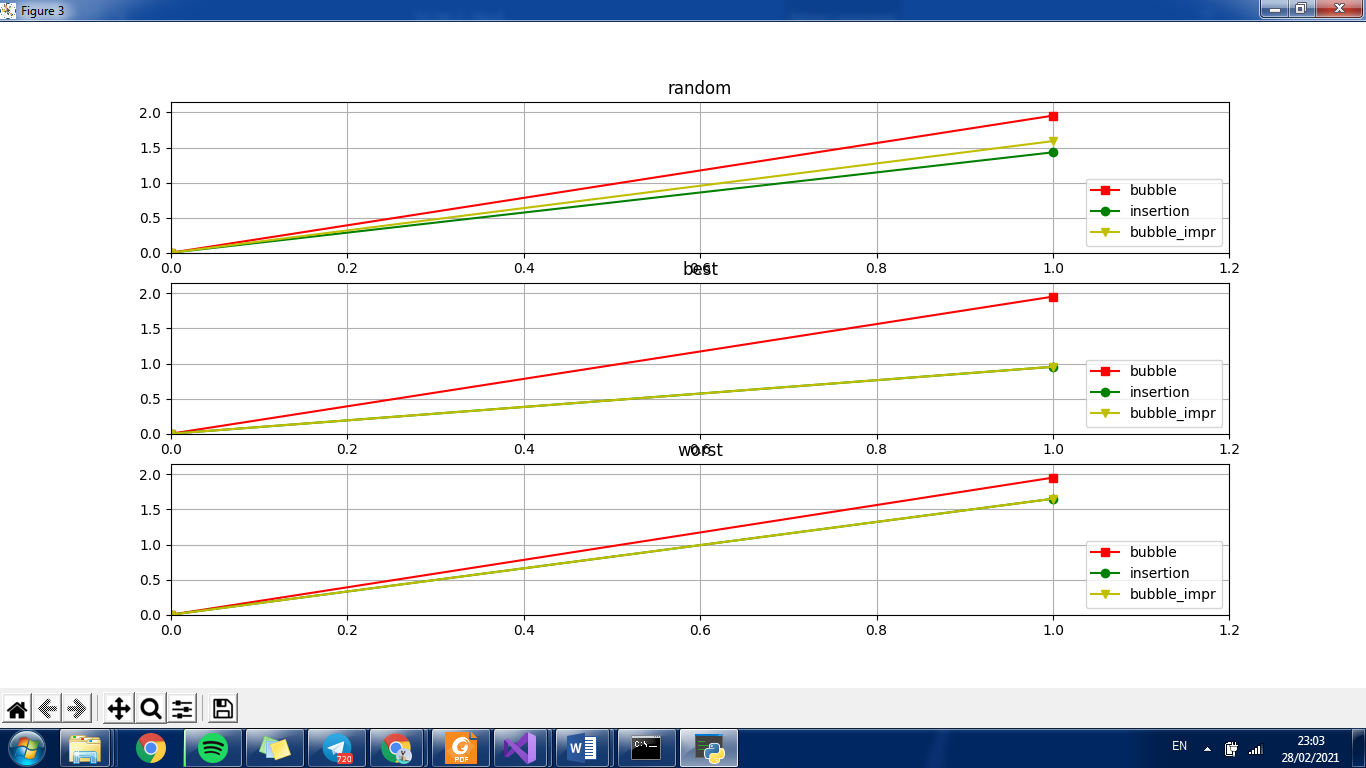
ax.set\_ylim( (0, y\_max\*1.1) )

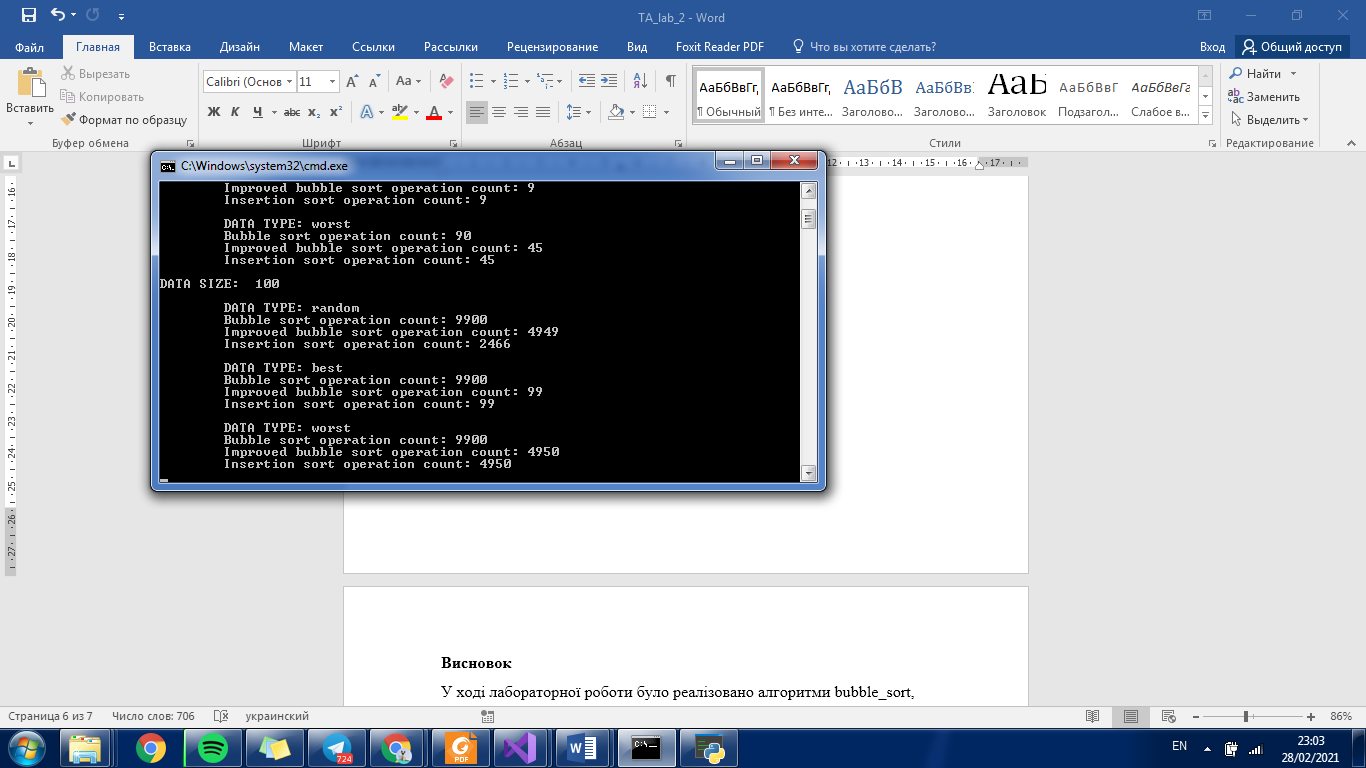
ax.legend(loc=4)

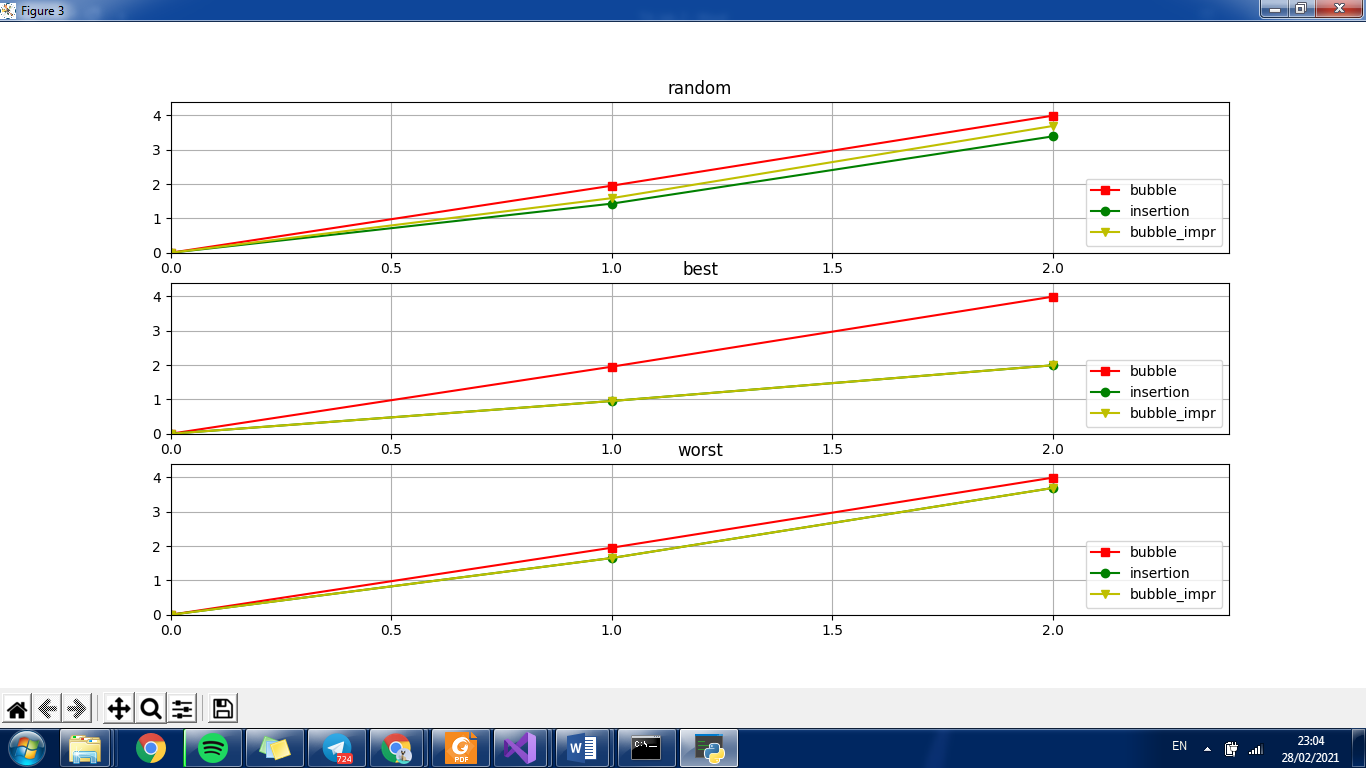
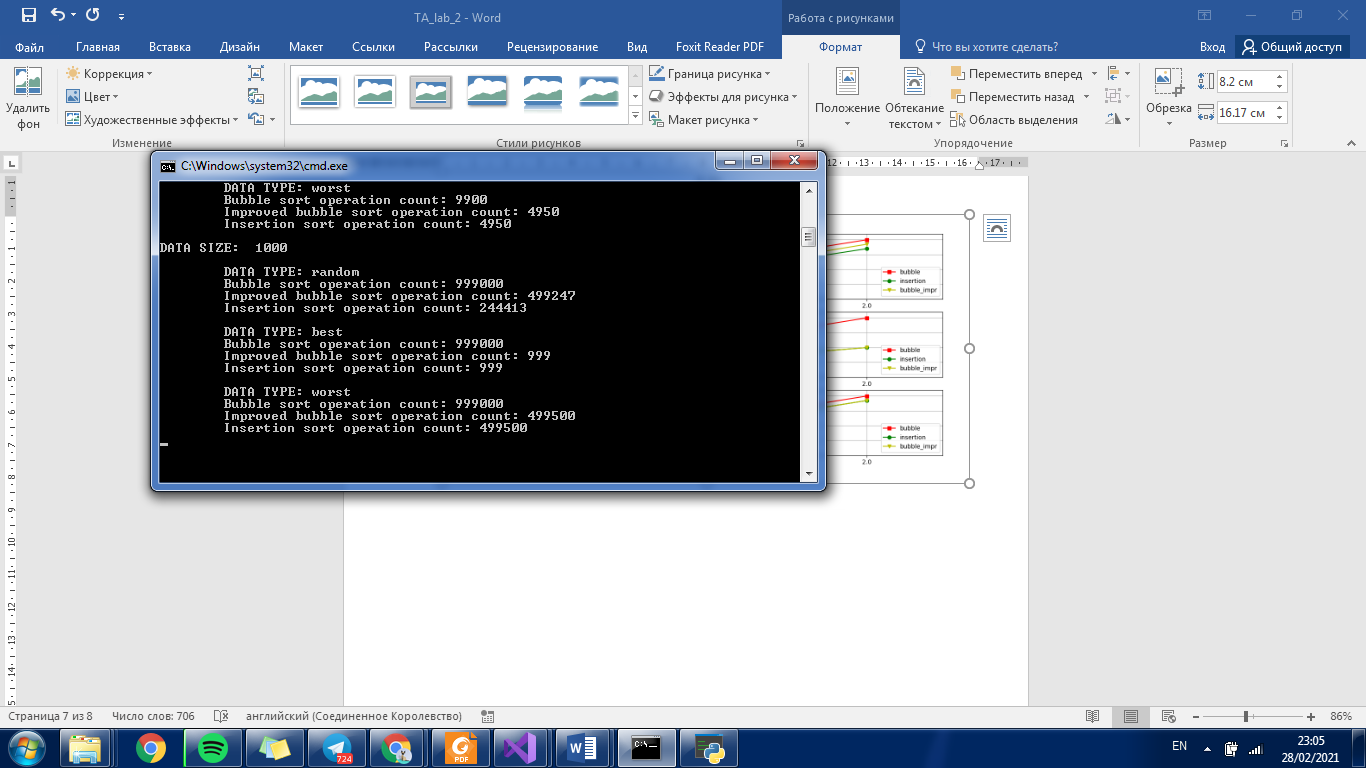
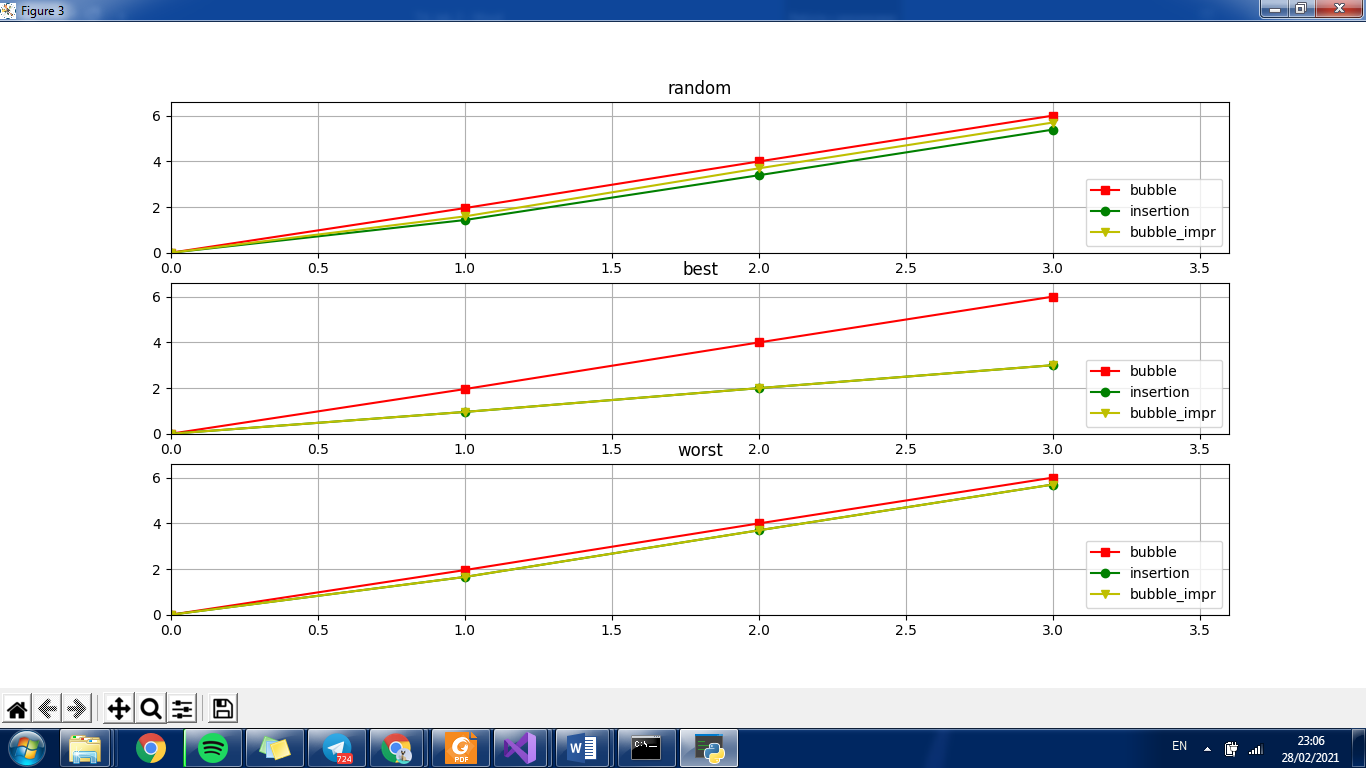
show()

**Результати**









**Висновок**

У ході лабораторної роботи було реалізовано алгоритми bubble\_sort, insertion\_sort та improved\_bubble\_sort. Хоча методи bubble\_sort та insertion\_sort мають однакову асимптотичну складність - О(n\*n), досліджено, що: у рандомному випадку на різних діапазонах значень метод включення працює швидше; у найгіршому та найкращому випадках метод включенням та вдосконалений бульбашковий метод прцюють однаково; звичайний бульбашковий метод працює скрізь повільніше.