**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

**ИНСТИТУТ ЦИФРЫ**

**ОТЧЁТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

«Рекурсия. Стратегия «разделяй и властвуй»»

Студентки 3 курса, ФИТ-211 группы

**Колесник Полины Олеговны**

Направление 02.03.02 – «Фундаментальная информатика и информационные технологии»

Руководитель:

Доцент Зимин А. И.

Работа защищена

« »

“ ” 2023 г.

Кемерово 2023 г.

**ОТЧЁТ О ПРОДЕЛАННОЙ РАБОТЕ**

**2 вариант**

**1 задание**

Реализуйте или найдите реализации двух сортировок: сортировка выбором и быстрая сортировка. Проведите вычислительные эксперименты и нарисуйте графики, показывающие зависимость времени выполнения двух алгоритмов от размера входных данных. Рассмотрите три варианта входных данных:

1. Список случайных чисел

2. Отсортированный список

3. Отсортированный в обратную сторону список

Для каждого из вариантов должен быть свой график зависимости.

import matplotlib.pyplot as plt

import random

import timeit

#1

def selection(arr):

    for i, e in enumerate(arr):

        mn = min(range(i, len(arr)), key=arr.\_\_getitem\_\_)

        arr[i], arr[mn] = arr[mn], e

    return arr

def quicksort(arr):

    if len(arr) < 2:

        return arr

    else:

        pivot = arr[0]

        less = [i for i in arr[1:] if i <= pivot]

        greater = [i for i in arr[1:] if i > pivot]

        return quicksort(less) + [pivot] + quicksort(greater)

plt\_x = []

time1 = []

time2 = []

for i in range(100, 1000, 100):

    plt\_x.append(i)

    lst\_random = [random.randint(-i, i) for \_ in range(i)]

    lst\_sorted = list(range(i))

    lst\_reverse\_sorted = reversed(list(range(i)))

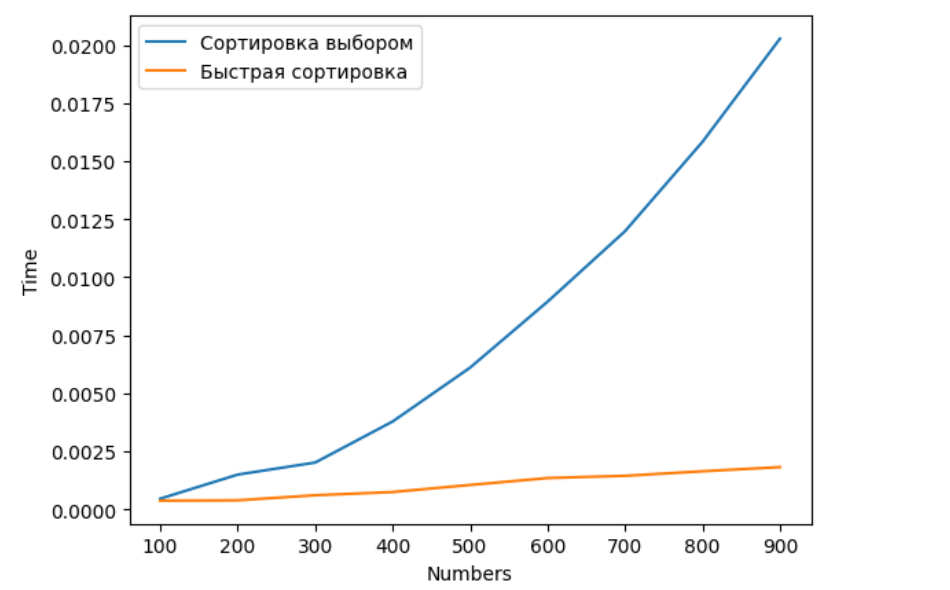
    #1

    time1.append(timeit.timeit(

        f"selection({lst\_random})", number=1, globals=globals()))

    time2.append(timeit.timeit(

        f"quicksort({lst\_random})", number=1, globals=globals()))



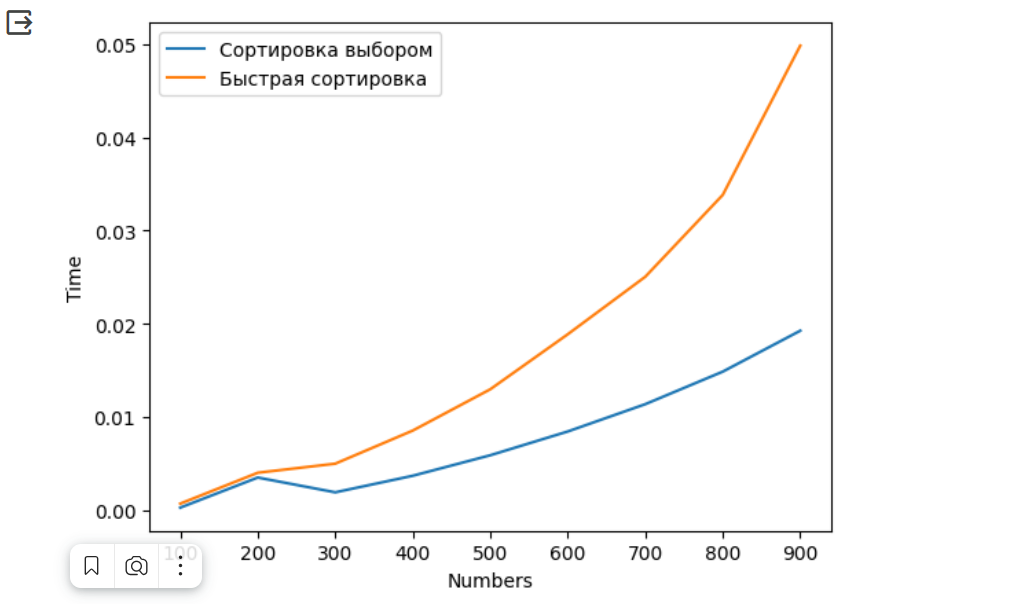
    #2

    time1.append(timeit.timeit(

        f"selection({lst\_sorted})", number=1, globals=globals()))

    time2.append(timeit.timeit(

        f"quicksort({lst\_sorted})", number=1, globals=globals()))



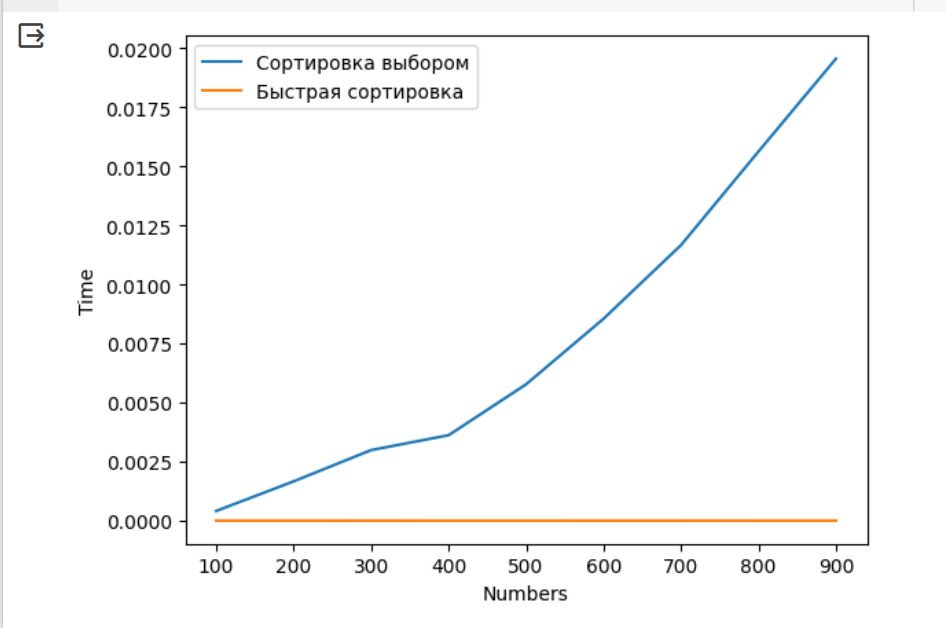
    #3

    time1.append(timeit.timeit(

        f"selection({[\*lst\_reverse\_sorted]})", number=1, globals=globals()))

    time2.append(timeit.timeit(

        f"quicksort({[\*lst\_reverse\_sorted]})", number=1, globals=globals()))



plt.xlabel('Numbers')

plt.ylabel('Time')

plt.plot(plt\_x, time1, label='Сортировка выбором')

plt.plot(plt\_x, time2, label='Быстрая сортировка')

plt.legend()

plt.show()

**2 задание**

Ряд Трибоначчи начинается с тройки 0, 0, 1, а каждое следующее число равно сумме трёх предыдущих. Числа нумеруются с 0.

Напишите функцию tribonacci(n), которая принимает в себя номер числа и возвращает n-ое число Трибоначчи. Функция должна быть рекурсивной.

При решении данной задачи не используйте циклы.

Укажите базовый и рекурсивный случаи функции.

# 2

def tribonacci(n):

    if n == 0 or n == 1: #базовый случай

        return 0

    elif n == 2:

        return 1

    else: # рекурсивный случай

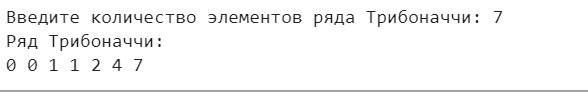
        return tribonacci(n - 1) + tribonacci(n - 2) + tribonacci(n - 3)

n = int(input("Введите количество элементов ряда Трибоначчи: "))

print("Ряд Трибоначчи:")

for i in range(n):

    print(tribonacci(i), end=' ')



**3 задание**

Для задачи вашего варианта напишите подходящую рекурсивную функцию.

При решении задачи не используйте циклы.

Укажите базовый и рекурсивный случаи вашей функции.

2 вариант

Напишите рекурсивную функцию recursive\_count(some\_list, t), которая возвращает количество вхождений элемента t в список some\_list. Рекурсивный вызов должен быть устроен так же, как в функции find\_max(A) из лекции.

# 3

def recursive\_count(some\_list, t):

    def rcount(lo, hi):

        if lo == hi: # базовый случай

            return 1 if some\_list[lo] == t else 0

        mid = (lo + hi) // 2 # рекурсивный случай

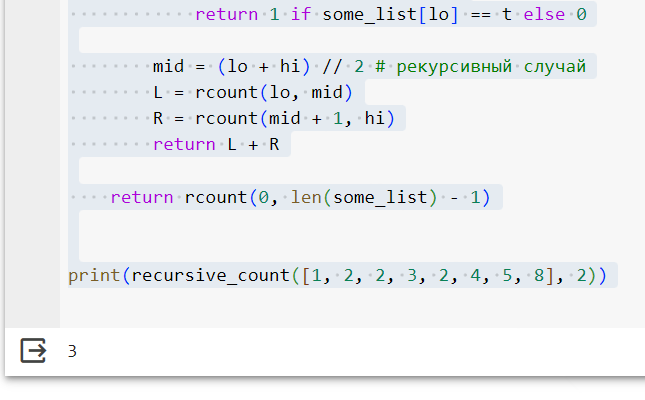
        L = rcount(lo, mid)

        R = rcount(mid + 1, hi)

        return L + R

    return rcount(0, len(some\_list) - 1)

print(recursive\_count([1, 2, 2, 3, 2, 4, 5, 8], 2))



**4 задание**

Числа Фибоначчи - это последовательность, которая вычисляется по рекуррентному соотношению FN = FN - 1 + FN - 2, где F0 = 0, а F1 = 1. Числа Люка задаются примерно так же: LN = LN - 1 + LN - 2, где L0 = 2, а L1 = 1. Пользуясь стандартным рекурсивным подходом, напишите две функции - fibonacci(n) и lucas(n), которые будут вычислять n-е число Фибоначчи и Люка соответственно. Измерьте время, которое требуется для вычисления FN и LN вплоть до N = 40. Если ваш компьютер достаточно быстрый, N стоит увеличить для повышения точности, а если слишком медленный - уменьшить, чтобы измерения заканчивались за приемлемое время.

Затем напишите функцию fib\_with\_lucas(n) и вспомогательную к ней, lucas\_with\_fib(n), пользуясь такими свойствами этих последовательностей:

• fib\_with\_lucas(n) – если поделить n нацело пополам и взять i = n // 2, а j = n - i, то Fn = Fi + j = (Fi + Lj) (Fj + Li) // 2;

• lucas\_with\_fib(n) – в свою очередь, LN = FN - 1+FN + 1.

Сравните быстродействие fibonacci() и fib\_with\_lucas(). Постройте график.

# 4

def fibonacci(n):

    if n == 1:

        return 1

    elif n == 0:

        return 0

    else:

        return fibonacci(n - 1) + fibonacci(n - 2)

def lucas(n):

    if n == 1:

        return 1

    elif n == 0:

        return 2

    else:

        return lucas(n - 1) + lucas(n - 2)

def fib\_with\_lucas(n):

    i = n // 2

    j = n - i

    return (fibonacci(i) + lucas(j)) \* (fibonacci(j) + lucas(i)) // 2

def lucas\_with\_fib(n):

    return fibonacci(n - 1) + fibonacci(n + 1)

plt\_x = []

time1 = []

time2 = []

time3 = []

time4 = []

for i in range(1, 30, 2):

    plt\_x.append(i)

    time1.append(timeit.timeit(

        f"fibonacci({i})", number=1, globals=globals()))

    time2.append(timeit.timeit(

        f"lucas({i})", number=1, globals=globals()))

    time3.append(timeit.timeit(

        f"fib\_with\_lucas({i})", number=1, globals=globals()))

    time4.append(timeit.timeit(

        f"lucas\_with\_fib({i})", number=1, globals=globals()))

plt.figure(figsize=(12, 5))

plt.subplot(1, 3, 1)

plt.xlabel('Numbers')

plt.ylabel('Time')

plt.plot(plt\_x, time1, label='Фибоначи')

plt.plot(plt\_x, time2, label='Числа Люка')

plt.legend()

plt.subplot(1, 3, 2)

plt.xlabel('Numbers')

plt.ylabel('Time')

plt.plot(plt\_x, time3, label='Фибоначи c Люкой')

plt.plot(plt\_x, time4, label='Люка с Фибоначи')

plt.legend()

plt.subplot(1, 3, 3)

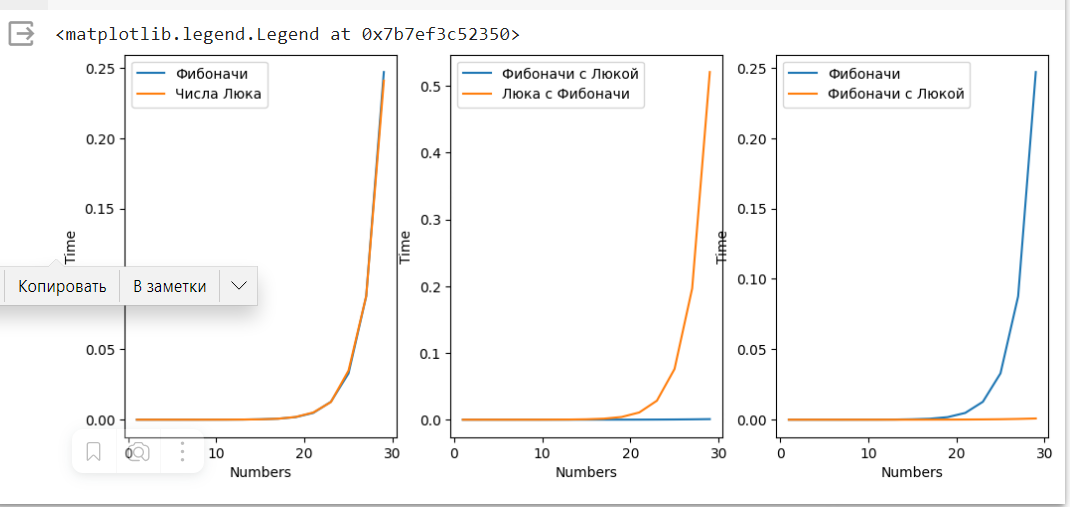
plt.xlabel('Numbers')

plt.ylabel('Time')

plt.plot(plt\_x, time1, label='Фибоначи')

plt.plot(plt\_x, time3, label='Фибоначи c Люкой')

plt.legend()



**5 задание**

Хорошим примером для иллюстрации рекурсивных алгоритмов являются задачи рисования фракталов. Далее представлена программа, которая при помощи модуля turtle (turtle — Turtle graphics — Python 3.9.7 documentation) рисует фрактальное дерево.

Запустите программу и разберитесь в ее коде. Попробуйте поменять параметры при вызове функции tree и внутри нее.

Последовательно измените программу для рисования рекурсивного дерева по следующим пунктам:

1. Измените толщину ветвей, чтобы при уменьшении branchLen линии становились тоньше.

2. Измените цвет ветвей таким образом, чтобы самые короткие ветви окрашивались как листья.

3. Измените угол поворота черепахи, чтобы каждая ветвь поворачивалась произвольным образом в некотором диапазоне. Например, выбирайте угол между 15-ю и 45-ю градусами. Поэкспериментируйте в поисках лучшего вида.

4. Измените рекурсивную часть branchLen, чтобы каждый раз вычиталось произвольное значение из некоторого диапазона вместо некой постоянной величины.

В отчете приведите несколько примеров получившихся деревьев.

# 5.1

import turtle

def tree(branchLen, t):

    if branchLen > 5:

        t.pensize(branchLen / 5)  # 1

        t.forward(branchLen)

        t.right(20)

        tree(branchLen - 15, t)

        t.left(40)

        tree(branchLen - 15, t)

        t.right(20)

        t.backward(branchLen)

def main():

    t = turtle.Turtle()

    myWin = turtle.Screen()

    t.left(90)

    t.up()

    t.backward(100)

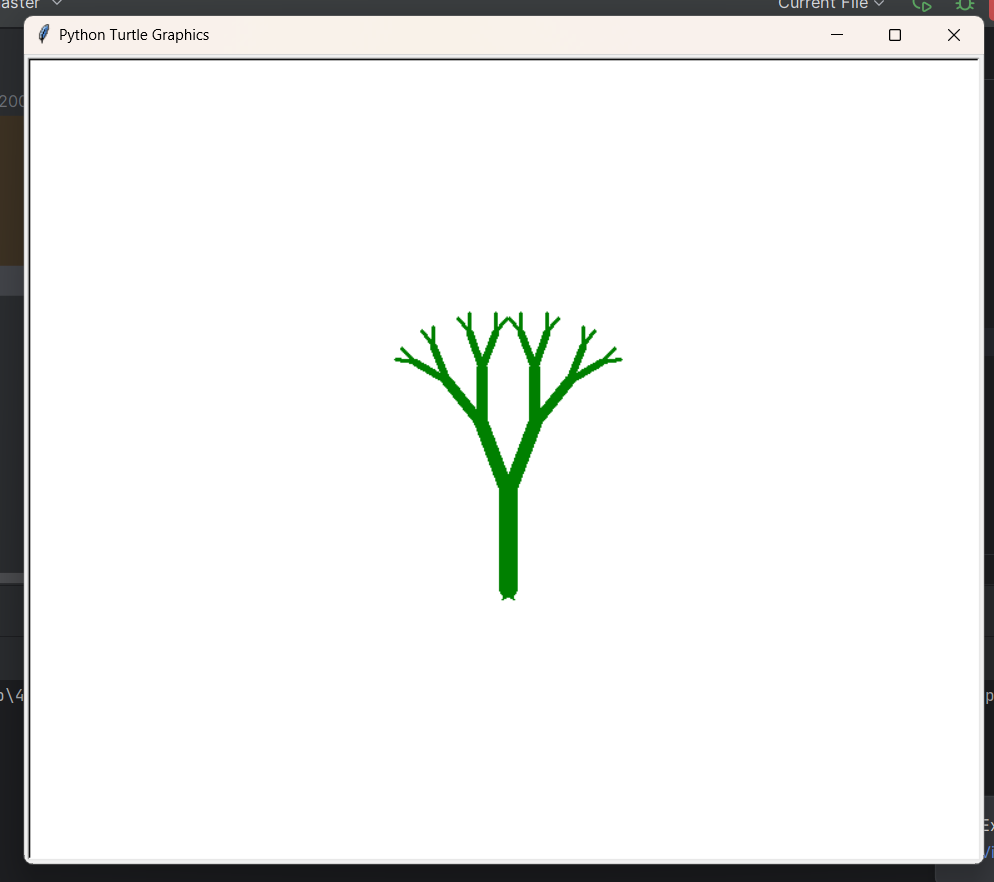
    t.down()

    t.color("green")

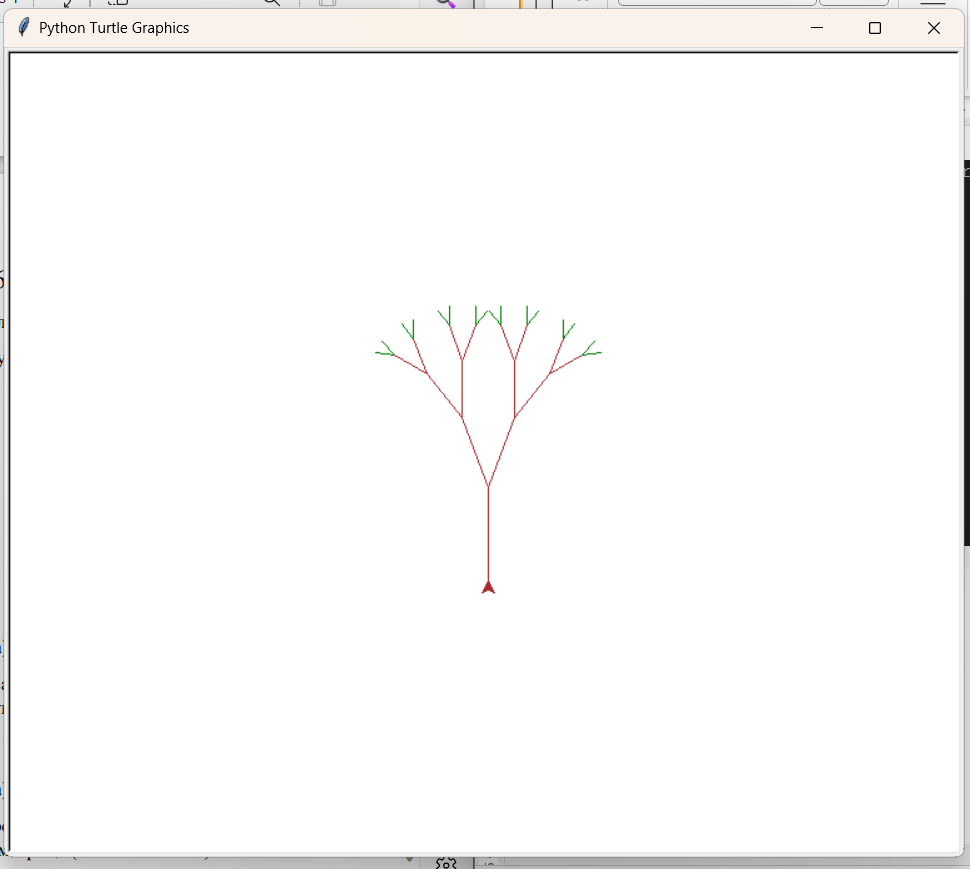
    tree(75, t)

    myWin.exitonclick()

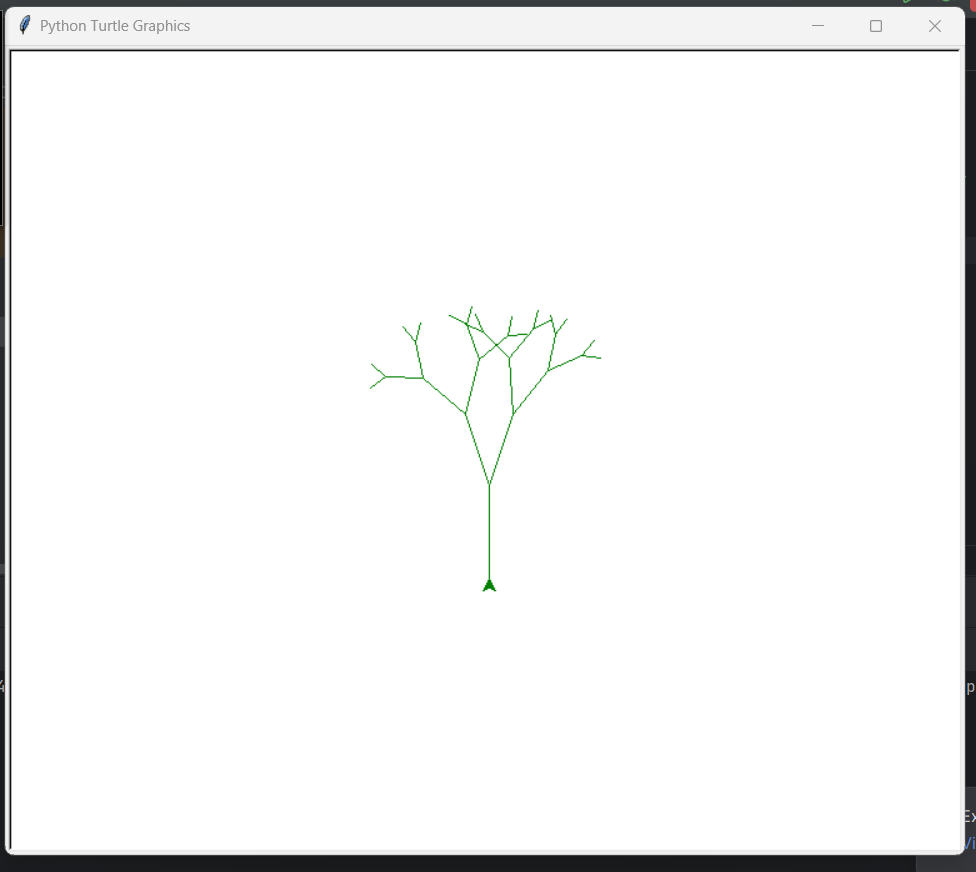
main()

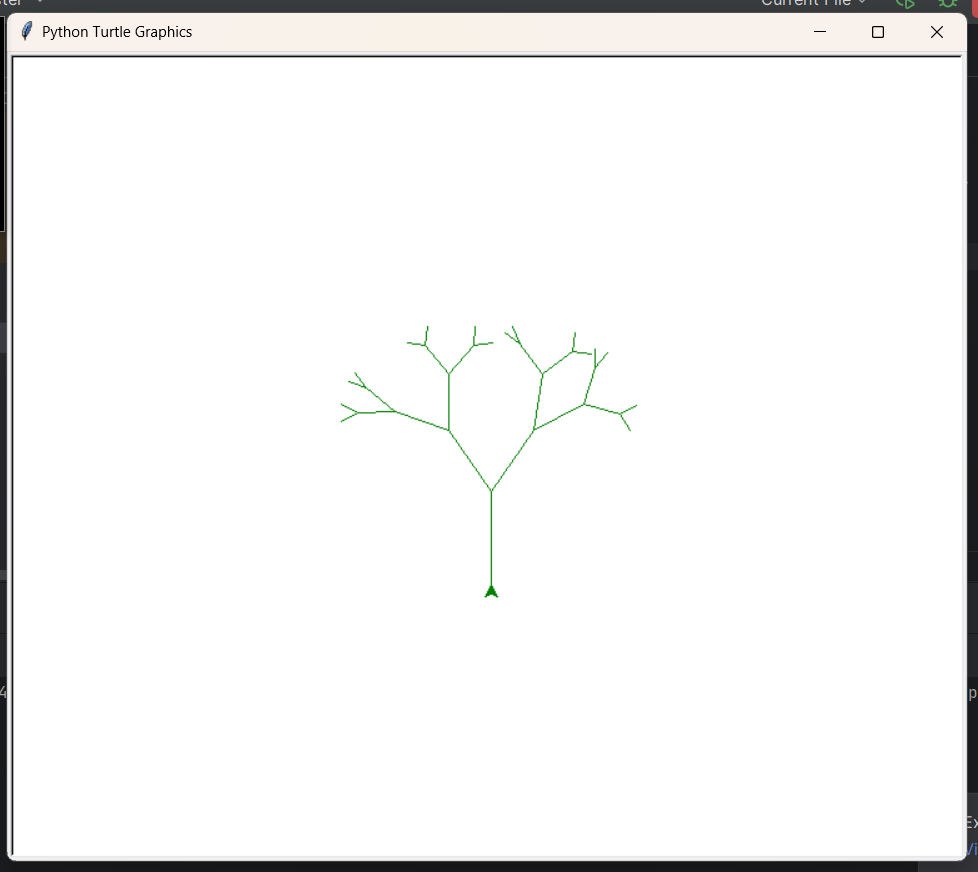


# 5.2  
import turtle  
  
  
# Задаем цвет для "листьев" и "дерева" для пункта 2  
def branch\_color(branchLen):  
 if branchLen > 15:  
 return "brown"  
 else:  
 return "green"  
  
  
def tree(branchLen, t):  
 if branchLen > 5:  
 t.color(branch\_color(branchLen)) # 2  
 t.forward(branchLen)  
 t.right(20)  
 tree(branchLen - 15, t)  
 t.left(40)  
 tree(branchLen - 15, t)  
 t.right(20)  
 t.color(branch\_color(branchLen)) # 2  
 t.backward(branchLen)  
  
  
def main():  
 t = turtle.Turtle()  
 myWin = turtle.Screen()  
 t.left(90)  
 t.up()  
 t.backward(100)  
 t.down()  
 t.color("green")  
 tree(75, t)  
 myWin.exitonclick()  
  
  
main()

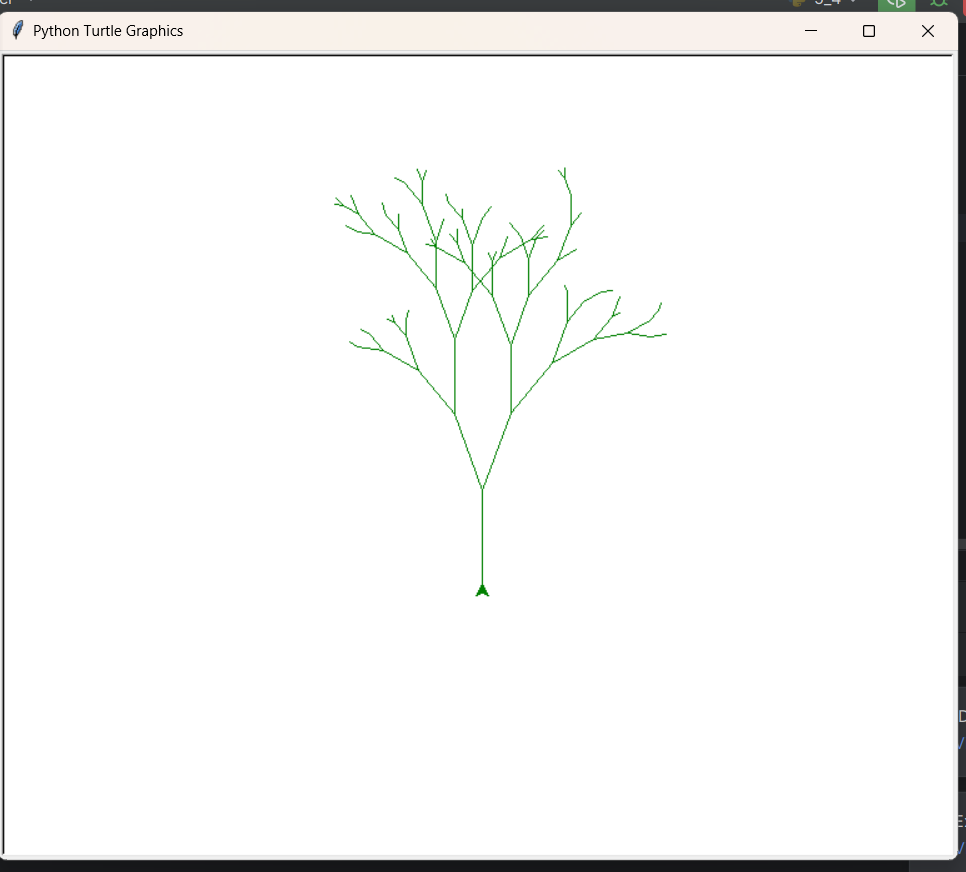


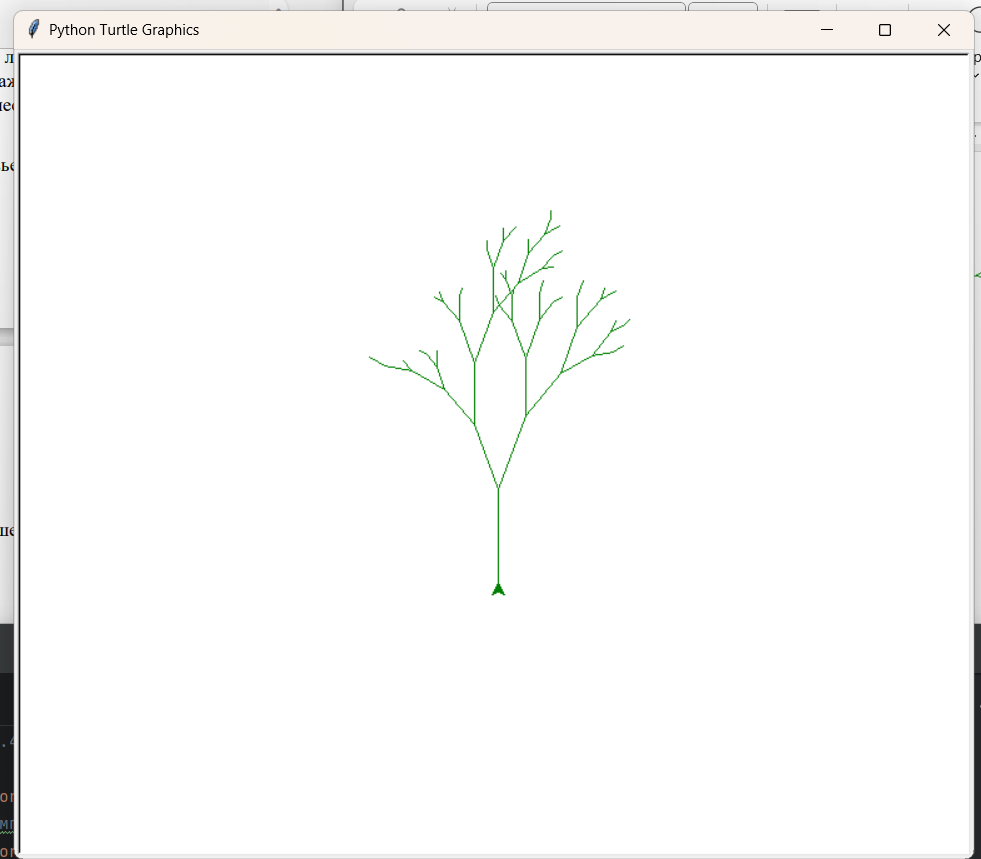
# 5.3  
  
import turtle  
# Импортируем рандом для произвольного выбора углов для пункта 3  
import random  
  
  
def tree(branchLen, t):  
 if branchLen > 5:  
 angle = random.randint(15, 45) # 3  
 t.forward(branchLen)  
 t.right(angle) # 3  
 tree(branchLen - 15, t)  
 t.left(angle \* 2) # 3  
 tree(branchLen - 15, t)  
 t.right(angle) # 3  
 t.backward(branchLen)  
  
  
def main():  
 t = turtle.Turtle()  
 myWin = turtle.Screen()  
 t.left(90)  
 t.up()  
 t.backward(100)  
 t.down()  
 t.color("green")  
 tree(75, t)  
 myWin.exitonclick()  
  
  
main()





# 5.4   
  
import turtle  
# Импортируем рандом для изменения рекурсивной части branchLen для пункта 4 (мной взять диапазон от 5 до 20)  
import random  
  
  
def tree(branchLen, t):  
 if branchLen > 5:  
 t.forward(branchLen)  
 t.right(20)  
 tree(branchLen - random.randint(5, 20), t) # 4  
 t.left(40)  
 tree(branchLen - random.randint(5, 20), t) # 4  
 t.right(20)  
 t.backward(branchLen)  
  
  
def main():  
 t = turtle.Turtle()  
 myWin = turtle.Screen()  
 t.left(90)  
 t.up()  
 t.backward(100)  
 t.down()  
 t.color("green")  
 tree(75, t)  
 myWin.exitonclick()  
  
  
main()



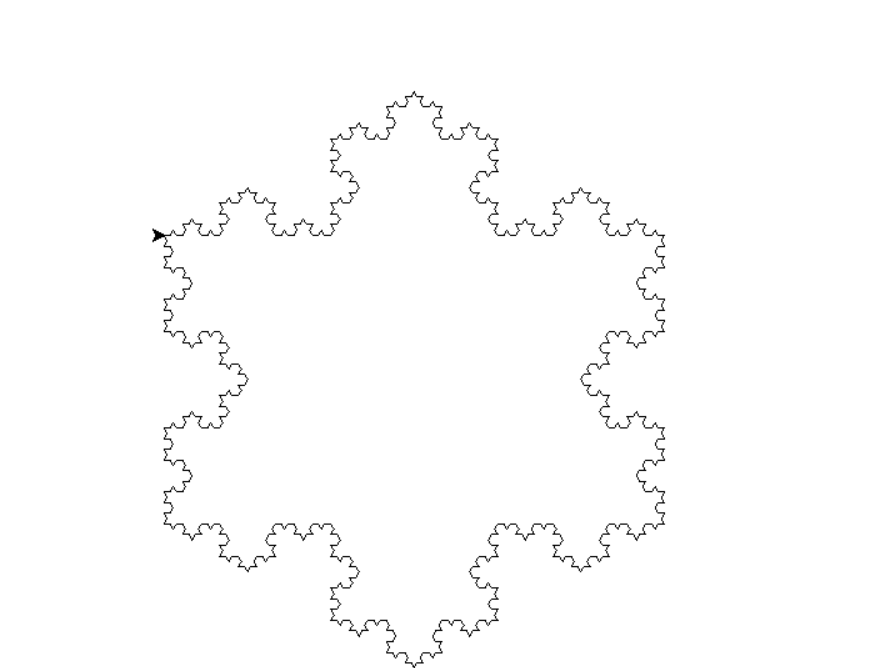


**6 задание**

При помощи модуля turtle нарисуйте фрактал, указанный в вашем варианте (материалы Рекурсия: фракталы (mipt-cs.github.io)).

Вариант 2. Снежинка Коха.

import turtle  
  
  
def snowflake(length, depth):  
 if depth == 0:  
 turtle.forward(length)  
 return  
 length /= 3.0  
 snowflake(length, depth - 1)  
 turtle.left(60)  
 snowflake(length, depth - 1)  
 turtle.right(120)  
 snowflake(length, depth - 1)  
 turtle.left(60)  
 snowflake(length, depth - 1)  
  
  
turtle.speed(0)  
turtle.up()  
turtle.goto(-200, 0)  
turtle.down()  
for i in range(3):  
 snowflake(400, 4)  
 turtle.right(120)  
  
turtle.done()

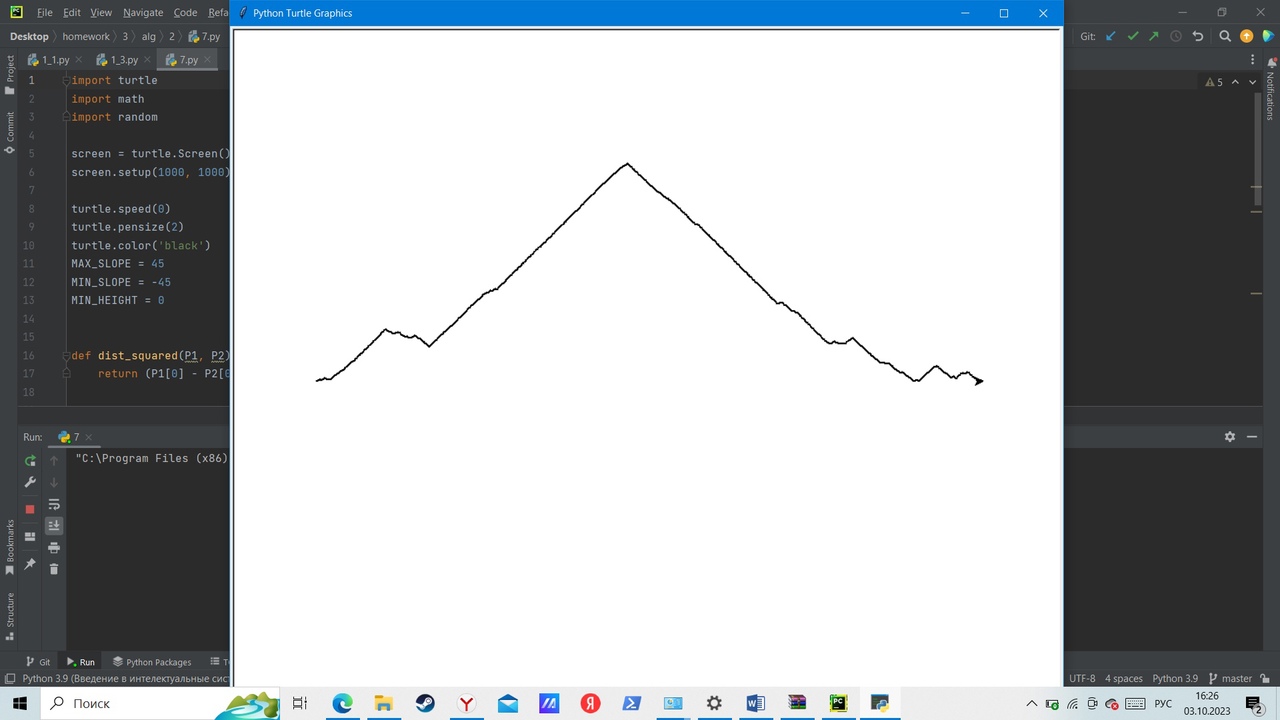


**7 задание**

Найдите или придумайте алгоритм для рисования фрактальных гор. Подсказка: одним из возможных методов будет использование треугольников.

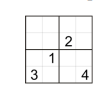
import turtle  
import math  
import random  
  
screen = turtle.Screen()  
screen.setup(1000, 1000)  
  
turtle.speed(0)  
turtle.pensize(2)  
turtle.color('black')  
MAX\_SLOPE = 45  
MIN\_SLOPE = -45  
MIN\_HEIGHT = 0  
  
  
def dist\_squared(P1, P2):  
 return (P1[0] - P2[0]) \*\* 2 + (P1[1] - P2[1]) \*\* 2  
  
  
def mountain(P1, P2):  
 if dist\_squared(P1, P2) < 9:  
 turtle.goto(P2)  
 return  
 x1, y1 = P1  
 x2, y2 = P2  
 x3 = random.uniform(x1, x2)  
 y3\_max = min((x3 - x1) \* math.tan(math.radians(MAX\_SLOPE)) + y1,  
 (x2 - x3) \* math.tan(-math.radians(MIN\_SLOPE)) + y2)  
 y3\_min = max((x3 - x1) \* math.tan(math.radians(MIN\_SLOPE)) + y1,  
 (x2 - x3) \* math.tan(-math.radians(MAX\_SLOPE)) + y2)  
 y3\_min = max(y3\_min, MIN\_HEIGHT)  
 y3 = random.uniform(y3\_min, y3\_max)  
 P3 = (x3, y3)  
 mountain(P1, P3)  
 mountain(P3, P2)  
 return  
  
  
turtle.up()  
  
turtle.goto(-400, MIN\_HEIGHT)  
turtle.down()  
mountain((-400, MIN\_HEIGHT), (400, MIN\_HEIGHT))  
turtle.done()





**8 задание**

Решите рекурсивно мини-судоку размером 4x4. Для этого напишите функцию solve\_sudoku(matrix), где matrix - целочисленная матрица (список списков).



В мини-судоку числа от 1 до 4 встречаются ровно один раз в каждой вертикали и горизонтали, а также в квадратах 2x2.

Укажите базовый и рекурсивный случаи вашего алгоритма.

Судоку представлено в виде таблицы чисел, в которой нулями обозначены пустые места:

0000

0020

0100

3004

Правильный ответ:

2341

1423

4132

3214

def solve\_sudoku(matrix):  
 # Поиск пустой клетки  
 for row in range(4):  
 for col in range(4):  
 if matrix[row][col] == 0:  
 for num in range(1, 5):  
 # Проверка возможных значений  
 if is\_valid(matrix, row, col, num):  
 matrix[row][col] = num  
 # Рекурсивный случай  
 if solve\_sudoku(matrix):  
 return True  
 # Отмена выбранного значения  
 matrix[row][col] = 0  
 return False  
 return True  
  
  
def is\_valid(matrix, row, col, num):  
 # Проверка строки и столбца  
 for i in range(4):  
 if matrix[row][i] == num or matrix[i][col] == num:  
 return False  
 # Проверка квадрата 2x2  
 start\_row = row // 2 \* 2  
 start\_col = col // 2 \* 2  
 for i in range(start\_row, start\_row + 2):  
 for j in range(start\_col, start\_col + 2):  
 if matrix[i][j] == num:  
 return False  
 return True  
  
  
matrix = [  
 [0, 0, 0, 0],  
 [0, 0, 2, 0],  
 [0, 1, 0, 0],  
 [3, 0, 0, 4]  
]  
  
if solve\_sudoku(matrix):  
 for row in matrix:  
 print(row)  
else:  
 print("Нет правильного решения")

