|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

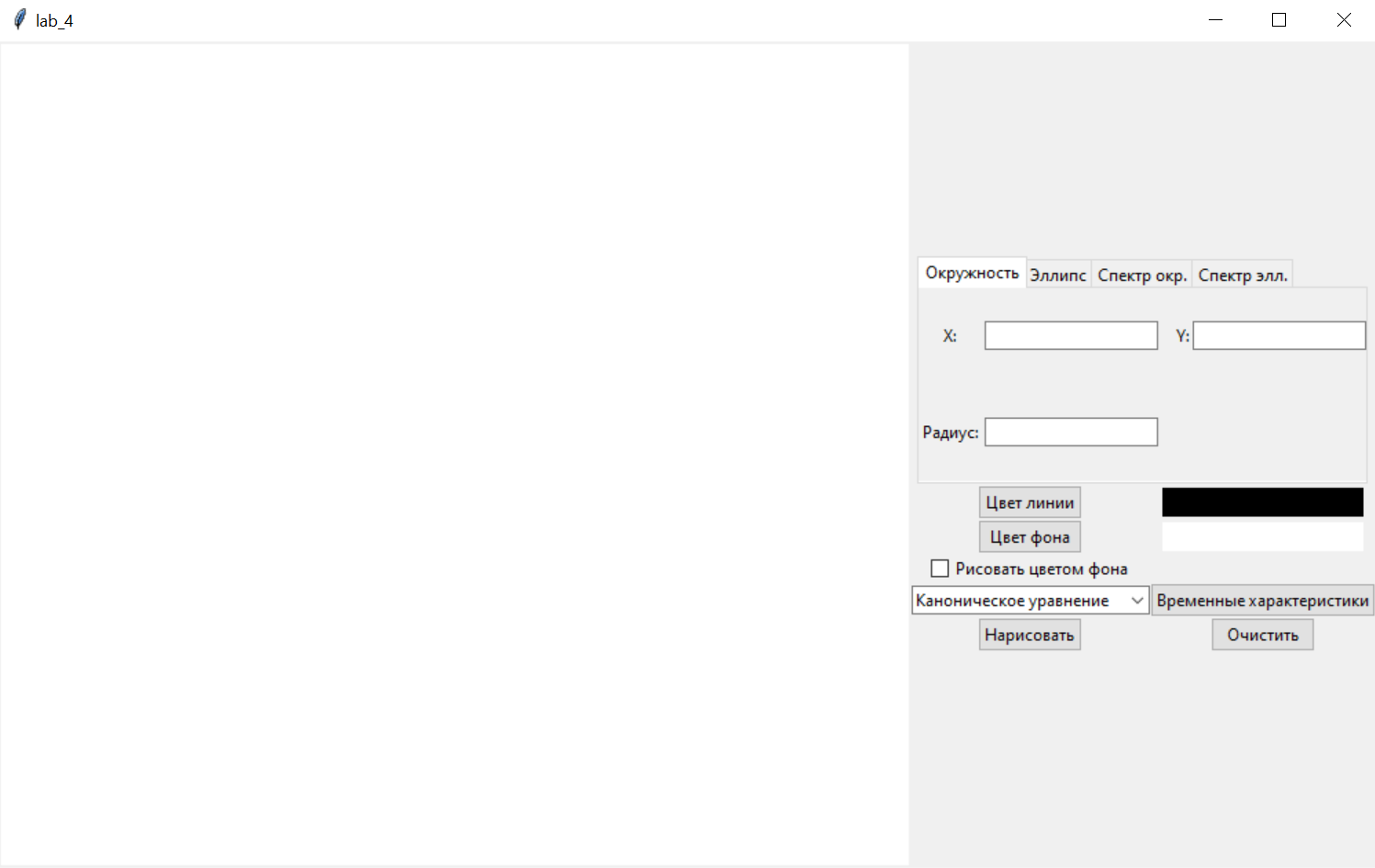
**Лабораторная работа № 4**

|  |  |
| --- | --- |
| **Тема: Реализация и исследование алгоритмов построения окружностей и эллипсов**  **Студент: Нгуен Ань Тхы**  **Группа: ИУ7-46Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель: Куров.А.В** |  |

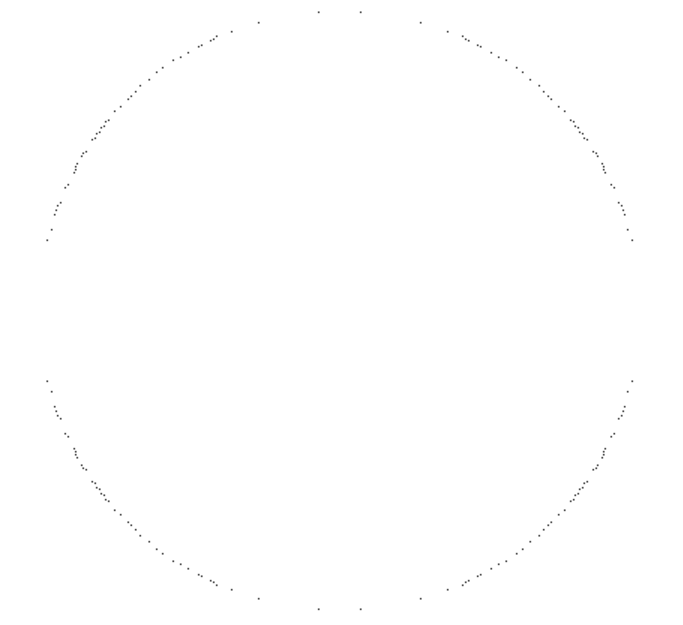
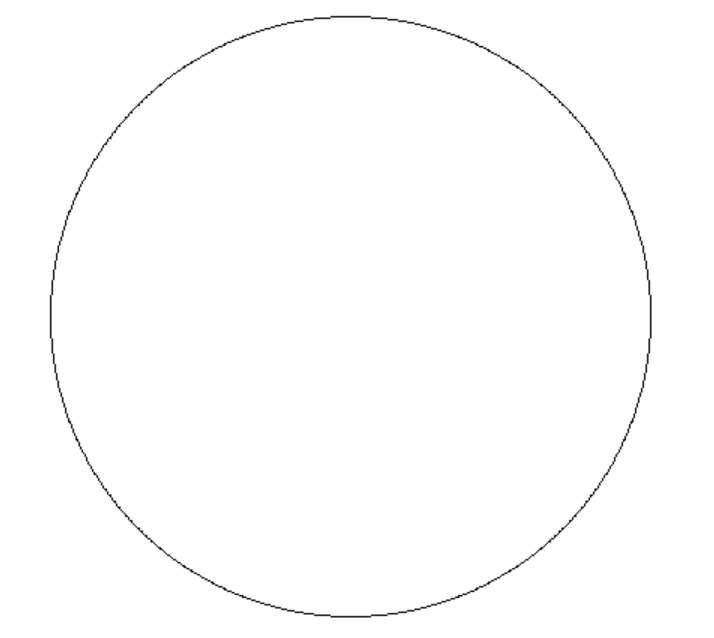
Москва.

2020 г.

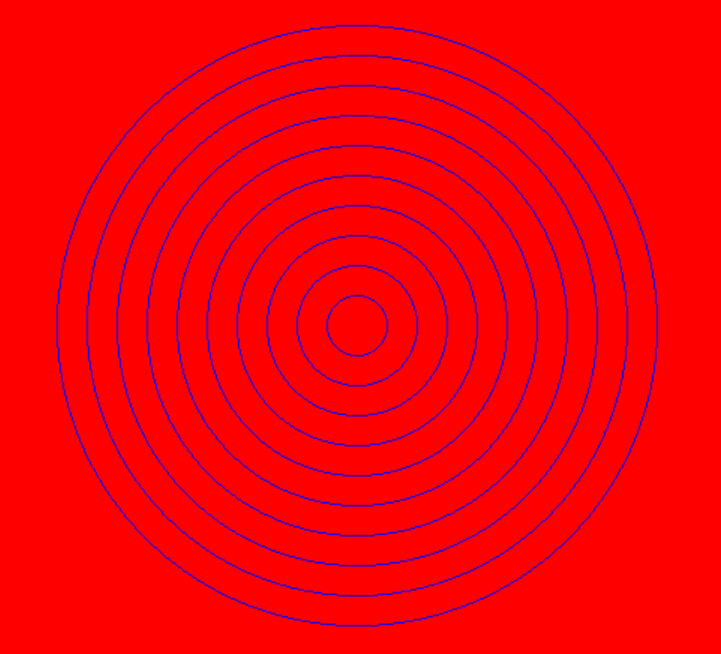
**Интерфей:**

****

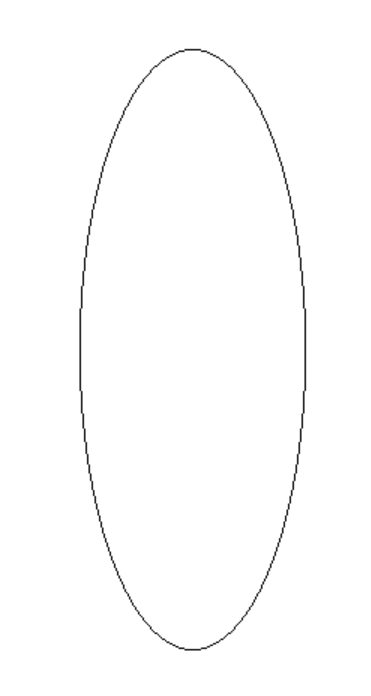
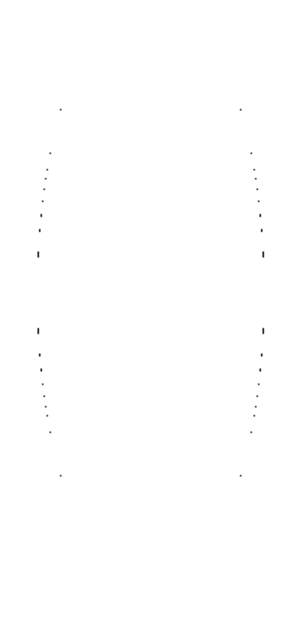
1. **Построение окружностей и сравнение их визуальных характеров:**



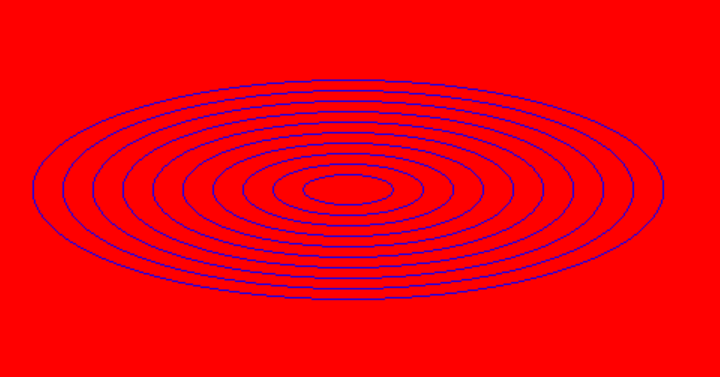
**Спектр окружностей:**

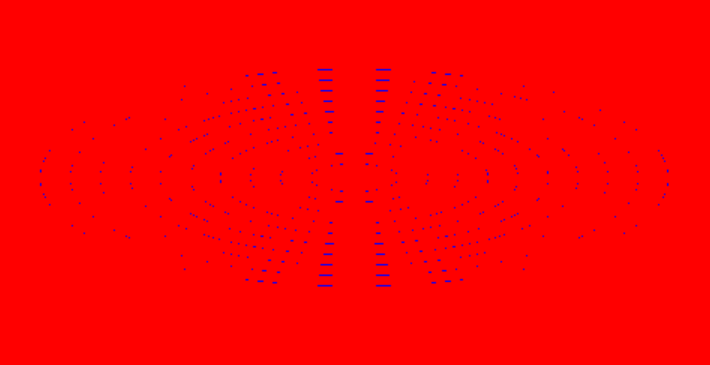
****

1. **Построение эллипсов и сравнение их визуальных характеров:**

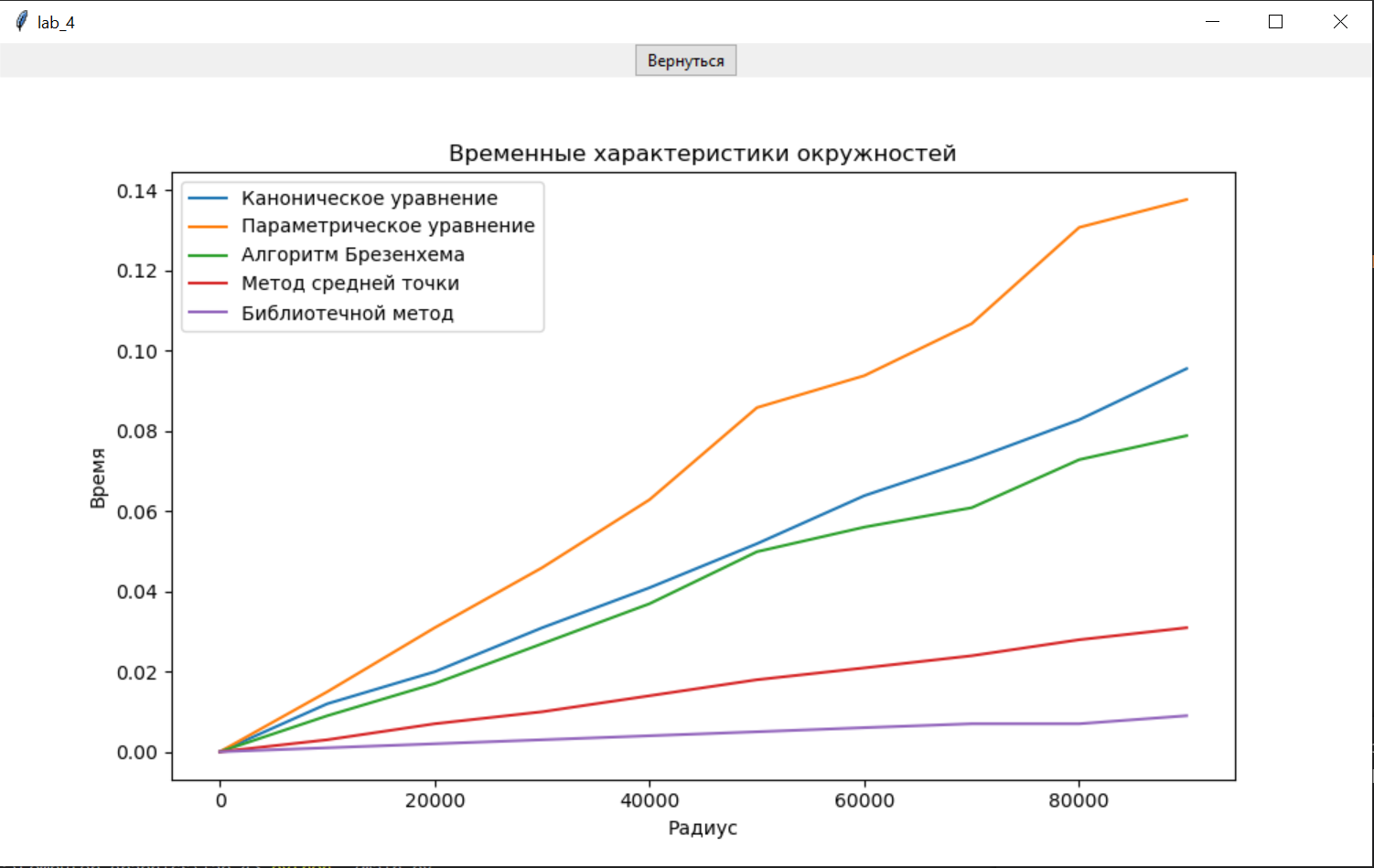
**** ****

**Спектр эллипсов:**

****

****

1. **Исследование временых характеристик:**

****

1. **Алгоритм:**
2. **Каноническое уравнение**
3. **Окружности:**

X2 + Y2 = R2 => Y = ±

Цикл построения окружностей (что x от 0 до R):

1. Y =
2. Высвечивание точки: (X, E(Y)), (X, E(-Y)), (-X, E(Y)), (-X, E(-Y))

Где E: операция округления до ближайщего целого значения

**б) Эллипсы:**

=> Y = ±B

Цикл построения Эллипса (что x от 0 до A):

i) Y = B

1. Высвечивание точки: (X, E(Y)), (X, E(-Y)), (-X, E(Y)), (-X, E(-Y))

Где E: операция округления до ближайщего целого значения

1. **Параметрическое уравнение**
2. **Окружности:**

X = X0 + R.cos(t)

Y = Y0 + R.sin(t), Где t [0, 2π)

L = πR/2

Цикл построения окружностей (что t от 0 до L):

1. X = R.cos(t / R), Y = R.sin(t / R)
2. Высвечивание точки: (E(X), E(Y)), (E(X), E(-Y)), (E(-X), E(Y)), (E(-X), E(-Y))

Где E: операция округления до ближайщего целого значения

**б) Эллипсы:**

X = X0 + A.cos(t)

Y = Y0 + B.sin(t), Где t [0, 2π)

M = max(A,B)

L = π \* M/2

Цикл построения Эллипса (что t от 0 до L):

1. X = A.cos(t / M), Y = B.sin(t / M)
2. Высвечивание точки: (E(X), E(Y)), (E(X), E(-Y)), (E(-X), E(Y)), (E(-X), E(-Y))

Где E: операция округления до ближайщего целого значения

1. **Алгоритм Брезенхема**
2. **Окружности:**
3. X = 0, Y = R, D = 2 ( 1 – R) (начальные значения)
4. Цикл построения окружностей: (пока Y >= 0)

2.1 Высвечивание точки (X, Y), (X, -Y), (-X, Y), (-X, -Y)

2.2 Если D < 0: (пиксель лежит внутри окружности)

i) X = X + 1

ii) D1 = 2 (D + Y) – 1

iii) Если D1 <= 0 (горизонтальный шаг): D = D + 2X + 1,

иначе (диагональный шаг): Y = Y – 1, D = D + 2(X – Y + 1)

2.3 Если D > 0: (пиксель лежит внеш окружности)

i) Y = Y - 1

ii) D2 = 2 (D - X) – 1

iii) Если D2 > 0 (вертикальный шаг): D = D – 2Y + 1,

иначе (диагональный шаг): X = X + 1, D = D + 2(X – Y + 1)

2.4 Если D = 0: (пиксель лежит на окружности)

i) X = X + 1

ii) Y = Y – 1

iii) D = D + 2 (X – Y + 1)

1. Конец

**б) Эллипы:**

1. X = 0, Y = B, D = 2 ( B2/2 – 2A2B+A2/2) (начальные значения)
2. Цикл построения эллипса: (пока Y >= 0)

2.1 Высвечивание точки (X, Y), (X, -Y), (-X, Y), (-X, -Y)

2.2 Если D < 0: (пиксель лежит внутри эллипса)

i) D1 = 2(D + A2Y) – A2

ii) X = X + 1

iii) Если D1 <= 0 (горизонтальный шаг): D = D + 2B2X + B2,

иначе (диагональный шаг): Y = Y – 1, D = D + 2(B2X – A2Y)+ A2+B2

2.3 Если D > 0: (пиксель лежит внеш окружности)

i) Y = Y - 1

ii) D2 = 2 (D – B2X) – B2

iii) Если D2> 0 (вертикальный шаг): D = D – 2A2Y + A2,

иначе (диагональный шаг): X = X + 1, D = D + 2(B2 X –A2Y)+A2+B2

2.4 Если D = 0: (пиксель лежит на окружности)

i) X = X + 1

ii) Y = Y – 1

iii) D = D + 2 (B2X – A2Y) + A2+B2

1. Конец
2. **Метод средней точки**
3. **Окружности:**
4. X = 0, Y = R, P = 5/4 – R (Начальные значения)
5. Цикл построения окружностей:

2.1 Высвечивание точки (X, Y), (X, -Y), (-X, Y), (-X, -Y), (Y, X), (Y, -X), (-Y, X), (-Y, -X)

2.2 X = X + 1

2.3 Если P < 0 (Средняя точка внутри окружности):

P = P + 2X + 1 (горизонтальный шаг)

Иначе (Средняя точка внеш окружности):

Y = Y - 1

P = P + 2(X – Y) + 1 (диагональный шаг)

* 1. Повторять цикл до тех пор, пока не получится X >= Y

3. Конец

**б)Эллипсы:**

1. X = 0, Y = B(Начальное положение)
2. P = B2 – A2B + 1/4A2 (начальное значение параметра принятия решения в области тангенса < 1)
3. Пока 2B2X < 2A2Y (пока тангенс угла наклона меньше 1)
   1. Высвечивание точки (X, Y), (X, -Y), (-X, Y), (-X, -Y)
   2. X = X + 1

2.3 Если P < 0 (Средняя точка внутри эллипса):

P = P + 2B2X + B2 (горизонтальный шаг)

Иначе (Средняя точка внеш окружности):

Y = Y - 1

P = P + 2B2X – 2A2Y + B2 (диагональный шаг)

4. P = B2(X+1/2)2 + A2(Y-1)2 – A2B2 (начальное значение параметра принятия решения в области тангенса >

5. Пока ( Y >= 0)

5.1 Высвечивание точки (X, Y), (X, -Y), (-X, Y), (-X, -Y)

5.2 Y = Y – 1

5.3 Если P > 0, то P = P – (2A2Y+A2)

Иначе P = P + 2B2X-2A2Y+A2

1. Конец
2. **Код программы:**

#Main

import tkinter as tk

import tkinter.ttk as ttk

from tkinter import messagebox

from tkinter.colorchooser import askcolor

# matplotlib to tkinter's canvas

import matplotlib

matplotlib.use("TkAgg")

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk

from matplotlib.figure import Figure

from typing import NamedTuple

from algorithms import \*

from testing import time\_test

# Circle struct

class Circle(NamedTuple):

    xc: int

    yc: int

    r: int

    color: str

    alg: int

# Ellipse struct

class Ellipse(NamedTuple):

    xc: int

    yc: int

    a: int

    b: int

    color: str

    alg: int

# App class

class Kg4App(tk.Tk):

    def \_\_init\_\_(self, \*args, \*\*kwargs):

        tk.Tk.\_\_init\_\_(self)

        options\_size = 336 # const

        can\_x = args[0] - options\_size

        can\_y = args[1]

        tk.Tk.title(self, "lab\_4")

        tk.Tk.geometry(self, str(can\_x + options\_size) + "x" + str(can\_y))

        container = tk.Frame(self)

        container.pack(side="top", fill="both", expand=True)

        container.grid\_rowconfigure(0, weight=1)

        container.grid\_columnconfigure(0, weight=1)

        self.frames = {}

        for F in (MainPage, GraphPage):

            frame = F(container, self)

            self.frames[F] = frame

            frame.grid(row=0, column=0, sticky="nsew")

        self.show\_frame(MainPage)

    def show\_frame(self, cont):

        frame = self.frames[cont]

        frame.tkraise()

# Main page class

class MainPage(tk.Frame):

    def \_\_init\_\_(self, parent, controller):

        tk.Frame.\_\_init\_\_(self, parent)

        self.color\_bg = "#fff"

        self.color\_line = "#000000"

        self.color\_pen = self.color\_line

        self.objects = []

        self.res = []

        # TABS

        self.work = tk.Frame(self)

        self.tab\_parent = ttk.Notebook(self.work, height=140)

        self.tab1 = ttk.Frame(self.tab\_parent)

        self.tab2 = ttk.Frame(self.tab\_parent)

        self.tab3 = ttk.Frame(self.tab\_parent)

        self.tab4 = ttk.Frame(self.tab\_parent)

        self.tab\_parent.add(self.tab1, text="Окружность")

        self.tab\_parent.add(self.tab2, text="Эллипс")

        self.tab\_parent.add(self.tab3, text="Спектр окр.")

        self.tab\_parent.add(self.tab4, text="Спектр элл.")

        self.tab\_parent.grid(row=0, column=0, columnspan=2)

        self.var1\_1 = tk.StringVar()

        self.var2\_1 = tk.StringVar()

        self.var3\_1 = tk.StringVar()

        self.var4\_1 = tk.StringVar()

        self.var1\_2 = tk.StringVar()

        self.var2\_2 = tk.StringVar()

        self.var3\_2 = tk.StringVar()

        self.var4\_2 = tk.StringVar()

        self.var1\_3 = tk.StringVar()

        self.var2\_3 = tk.StringVar()

        self.var3\_3 = tk.StringVar()

        self.var4\_3 = tk.StringVar()

        self.var1\_4 = tk.StringVar()

        self.var2\_4 = tk.StringVar()

        self.var3\_4 = tk.StringVar()

        self.var4\_4 = tk.StringVar()

        # Tab 1

        self.e\_w = 20

        self.tab1\_label\_1 = ttk.Label(self.tab1, text="X:").grid(row=0, column=0)

        self.tab1\_label\_2 = ttk.Label(self.tab1, text="   Y:").grid(row=0, column=2)

        self.tab1\_label\_3 = ttk.Label(self.tab1, text="Радиус:").grid(row=1, column=0)

        self.tab1\_entry\_x = ttk.Entry(self.tab1, textvariable=self.var1\_1, width=self.e\_w).grid(row=0, column=1)

        self.tab1\_entry\_y = ttk.Entry(self.tab1, textvariable=self.var2\_1, width=self.e\_w).grid(row=0, column=3)

        self.tab1\_entry\_r = ttk.Entry(self.tab1, textvariable=self.var3\_1, width=self.e\_w).grid(row=1, column=1)

        self.tab1.grid\_rowconfigure(0, weight=1)

        self.tab1.grid\_rowconfigure(1, weight=1)

        self.tab1.grid\_columnconfigure(0, weight=1)

        self.tab1.grid\_columnconfigure(1, weight=1)

        # Tab 2

        self.tab2\_label\_1 = ttk.Label(self.tab2, text="X:").grid(row=0, column=0)

        self.tab2\_label\_2 = ttk.Label(self.tab2, text="   Y:").grid(row=0, column=2)

        self.tab2\_label\_3 = ttk.Label(self.tab2, text="A:").grid(row=1, column=0)

        self.tab2\_label\_4 = ttk.Label(self.tab2, text="   B:").grid(row=1, column=2)

        self.tab2\_entry\_x = ttk.Entry(self.tab2, textvariable=self.var1\_2, width=self.e\_w).grid(row=0, column=1)

        self.tab2\_entry\_y = ttk.Entry(self.tab2, textvariable=self.var2\_2, width=self.e\_w).grid(row=0, column=3)

        self.tab2\_entry\_a = ttk.Entry(self.tab2, textvariable=self.var3\_2, width=self.e\_w).grid(row=1, column=1)

        self.tab2\_entry\_b = ttk.Entry(self.tab2, textvariable=self.var4\_2, width=self.e\_w).grid(row=1, column=3)

        self.tab2.grid\_rowconfigure(0, weight=1)

        self.tab2.grid\_rowconfigure(1, weight=1)

        self.tab2.grid\_columnconfigure(0, weight=1)

        self.tab2.grid\_columnconfigure(1, weight=1)

        # Tab 3

        self.e\_w = 10

        self.tab3\_label\_1 = ttk.Label(self.tab3, text="Начальный радиус:").grid(row=0, column=0)

        self.tab3\_label\_2 = ttk.Label(self.tab3, text="Шаг:").grid(row=1, column=0)

        self.tab3\_label\_3 = ttk.Label(self.tab3, text="   Количество:").grid(row=1, column=2)

        self.tab3\_entry\_r = ttk.Entry(self.tab3, textvariable=self.var1\_3, width=self.e\_w).grid(row=0, column=1)

        self.tab3\_entry\_step = ttk.Entry(self.tab3, textvariable=self.var2\_3, width=self.e\_w).grid(row=1, column=1)

        self.tab3\_entry\_amount = ttk.Entry(self.tab3, textvariable=self.var3\_3, width=self.e\_w).grid(row=1, column=3)

        self.tab3.grid\_rowconfigure(0, weight=1)

        self.tab3.grid\_rowconfigure(1, weight=1)

        self.tab3.grid\_columnconfigure(0, weight=1)

        self.tab3.grid\_columnconfigure(1, weight=1)

        # Tab 4

        self.tab4\_label\_1 = ttk.Label(self.tab4, text="Начальное A:").grid(row=0, column=0)

        self.tab4\_label\_2 = ttk.Label(self.tab4, text="   Начальное B:").grid(row=0, column=2)

        self.tab4\_label\_3 = ttk.Label(self.tab4, text="Шаг по A:").grid(row=1, column=0)

        self.tab4\_label\_4 = ttk.Label(self.tab4, text="   Количество:").grid(row=1, column=2)

        self.tab4\_entry\_x = ttk.Entry(self.tab4, textvariable=self.var1\_4, width=self.e\_w).grid(row=0, column=1)

        self.tab4\_entry\_y = ttk.Entry(self.tab4, textvariable=self.var2\_4, width=self.e\_w).grid(row=0, column=3)

        self.tab4\_entry\_a = ttk.Entry(self.tab4, textvariable=self.var3\_4, width=self.e\_w).grid(row=1, column=1)

        self.tab4\_entry\_b = ttk.Entry(self.tab4, textvariable=self.var4\_4, width=self.e\_w).grid(row=1, column=3)

        self.tab4.grid\_rowconfigure(0, weight=1)

        self.tab4.grid\_rowconfigure(1, weight=1)

        self.tab4.grid\_columnconfigure(0, weight=1)

        self.tab4.grid\_columnconfigure(1, weight=1)

        self.button\_line\_color = ttk.Button(self.work, text="Цвет линии",

                                            command=lambda: get\_color\_line(self)).grid(row=1, column=0)

        self.label\_line\_color = tk.Label(self.work, bg=self.color\_line, width=20)

        self.label\_line\_color.grid(row=1, column=1)

        self.button\_bg\_color = ttk.Button(self.work, text="Цвет фона",

                                          command=lambda: get\_color\_bg(self)).grid(row=2, column=0)

        self.label\_bg\_color = tk.Label(self.work, bg=self.color\_bg, width=20)

        self.label\_bg\_color.grid(row=2, column=1)

        self.ch\_var = tk.IntVar()

        self.ch\_button\_color = ttk.Checkbutton(self.work, text="Рисовать цветом фона", variable=self.ch\_var)

        self.ch\_var.set(0)

        self.ch\_button\_color.grid(row=3, column=0)

        self.list\_alg = ["Каноническое уравнение", "Параметрическое уравнение", "Алгоритм Брезенхема",

                    "Метод средней точки", "Библиотечный метод"]

        self.combobox\_alg = ttk.Combobox(self.work, width=25, values=self.list\_alg, state="readonly")

        self.combobox\_alg.current(0)

        self.combobox\_alg.grid(row=4, column=0)

        self.button\_test = ttk.Button(self.work, text="Временные характеристики",

                                 command=lambda: controller.show\_frame(GraphPage)).grid(row=4, column=1)

        self.button\_draw = ttk.Button(self.work, text="Нарисовать",

                                 command=lambda: draw(self)).grid(row=5, column=0)

        self.button\_clear = ttk.Button(self.work, text="Очистить",

                                  command=lambda: clear(self) ).grid(row=5, column=1)

        self.work.pack(side=tk.RIGHT)

        self.canvas = tk.Canvas(self, bg=self.color\_bg)

        self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand = True)

# Graph page class

class GraphPage(tk.Frame):

    def \_\_init\_\_(self, parent, controller):

        tk.Frame.\_\_init\_\_(self, parent)

        button1 = ttk.Button(self, text="Вернуться",

                             command=lambda: controller.show\_frame(MainPage))

        button1.pack()

        res = time\_test(100000)

        f = Figure(figsize=(5, 5), dpi=100)

        plt = f.add\_subplot(111)

        plt.plot(res[5], res[0], label="Каноническое уравнение")

        plt.plot(res[5], res[1], label="Параметрическое уравнение")

        plt.plot(res[5], res[2], label="Алгоритм Брезенхема")

        plt.plot(res[5], res[3], label="Метод средней точки")

        plt.plot(res[5], res[4], label ="Библиотечной метод")

        plt.set\_xlabel("Радиус")

        plt.set\_ylabel("Время")

        plt.set\_title("Временные характеристики окружностей")

        plt.legend()

        canvas = FigureCanvasTkAgg(f, self)

        canvas.draw()

        canvas.get\_tk\_widget().pack(side=tk.TOP, fill=tk.BOTH, expand = True)

# Creates window with warning message

def mes(text):

    messagebox.showinfo("Внимание", text)

# Get color for line

def get\_color\_line(self):

    color = askcolor()[1]

    if color:

        self.color\_line = color

        self.label\_line\_color.configure(bg=color)

# Get color for background and change it instantly

def get\_color\_bg(self):

    color = askcolor()[1]

    if color:

        self.color\_bg = color

        self.label\_bg\_color.configure(bg=color)

        self.canvas.configure(bg=self.color\_bg)

# Clear canvas

def clear(self):

    self.canvas.delete("all")

    self.canvas.configure(bg=self.color\_bg)

# Get data and create object

def draw(self):

    tab = self.tab\_parent.index(self.tab\_parent.select())

    # Получаем данные

    if tab == 0:

        try:

            xc = int(self.var1\_1.get())

            yc = int(self.var2\_1.get())

            r = int(self.var3\_1.get())

        except ValueError:

            mes("Неверные данные!")

            return -1

    elif tab == 1:

        try:

            xc = int(self.var1\_2.get())

            yc = int(self.var2\_2.get())

            a = int(self.var3\_2.get())

            b = int(self.var4\_2.get())

        except ValueError:

            mes("Неверные данные!")

            return -2

    elif tab == 2:

        try:

            r\_beg = int(self.var1\_3.get())

            step = int(self.var2\_3.get())

            amount = int(self.var3\_3.get())

        except ValueError:

            mes("Неверные данные!")

            return -3

    elif tab == 3:

        try:

            a\_beg = int(self.var1\_4.get())

            b\_beg = int(self.var2\_4.get())

            a\_step = int(self.var3\_4.get())

            amount = int(self.var4\_4.get())

        except ValueError:

            mes("Неверные данные!")

            return -4

    if int(self.ch\_var.get()) == 1:

        color = self.color\_bg

    else:

        color = self.color\_line

    alg = self.combobox\_alg.current()

    can\_x = self.canvas.winfo\_width()

    can\_y = self.canvas.winfo\_height()

    if tab == 0:

        circle = Circle(xc, yc, r, color, alg)

        draw\_func(self, circle)

    elif tab == 1:

        ellipse = Ellipse(xc, yc, a, b, color, alg)

        draw\_func(self, ellipse)

    elif tab == 2:

        for i in range(amount):

            circle = Circle(can\_x // 2, can\_y // 2, r\_beg, color, alg)

            draw\_func(self, circle)

            r\_beg += step

    elif tab == 3:

        b\_step = round(b\_beg / a\_beg \* a\_step)

        for i in range(amount):

            ellipse = Ellipse(can\_x // 2, can\_y // 2, a\_beg, b\_beg, color, alg)

            draw\_func(self, ellipse)

            a\_beg += a\_step

            b\_beg += b\_step

    for o in self.objects:

        draw\_func(self, o)

# Draw obj with algorithms

def draw\_func(self, obj):

    self.color\_pen = obj.color

    if type(obj) == Circle:

        if obj.alg == 0:

            draw\_circle\_canon(self, obj.xc, obj.yc, obj.r)

        elif obj.alg == 1:

            draw\_circle\_param(self, obj.xc, obj.yc, obj.r)

        elif obj.alg == 2:

            draw\_circle\_br(self, obj.xc, obj.yc, obj.r)

        elif obj.alg == 3:

            draw\_circle\_mid(self, obj.xc, obj.yc, obj.r)

        elif obj.alg == 4:

            draw\_circle\_std(self, obj.xc, obj.yc, obj.r)

    elif type(obj) == Ellipse:

        if obj.alg == 0:

            draw\_ellipse\_canon(self, obj.xc, obj.yc, obj.a, obj.b)

        elif obj.alg == 1:

            draw\_ellipse\_param(self, obj.xc, obj.yc, obj.a, obj.b)

        elif obj.alg == 2:

            draw\_ellipse\_br(self, obj.xc, obj.yc, obj.a, obj.b)

        elif obj.alg == 3:

            draw\_ellipse\_mid(self, obj.xc, obj.yc, obj.a, obj.b)

        elif obj.alg == 4:

            draw\_ellipse\_std(self, obj.xc, obj.yc, obj.a, obj.b)

if \_\_name\_\_ == '\_\_main\_\_':

    app = Kg4App(1000, 600)

    app.mainloop()

#Algorithms

from math import cos, sin, pi, sqrt

# Place pix on x, y

def draw\_pix(self, x, y):

    self.canvas.create\_line(x, y, x+1, y+1, fill=self.color\_pen)

    return 0

# ==========CIRCLES==========

def draw\_circle\_canon(self, xc, yc, r):

    for x in range(0, r + 1, 1):

        y = round(sqrt(r \*\* 2 - x \*\* 2))

        draw\_pix(self, xc + x, yc + y)

        draw\_pix(self, xc + x, yc - y)

        draw\_pix(self, xc - x, yc + y)

        draw\_pix(self, xc - x, yc - y)

    return 0

def draw\_circle\_param(self, xc, yc, r):

    len = round(pi \* r / 2)

    for i in range(0, len + 1, 1):

        x = round(r \* cos(i / r))

        y = round(r \* sin(i / r))

        draw\_pix(self, xc + x, yc + y)

        draw\_pix(self, xc + x, yc - y)

        draw\_pix(self, xc - x, yc + y)

        draw\_pix(self, xc - x, yc - y)

    return 0

def draw\_circle\_br(self, xc, yc, r):

    x = 0  # задание начальных значений

    y = r

    d = 2 \* (1 - r)  # значение D(x,y)  при (0,R)

    while y >= 0:

        # высвечивание текущего пиксела

        draw\_pix(self, xc + x, yc + y)

        draw\_pix(self, xc + x, yc - y)

        draw\_pix(self, xc - x, yc + y)

        draw\_pix(self, xc - x, yc - y)

        if d < 0:  # пиксель лежит внутри окружности

            d1 = 2 \* (d + y) - 1

            x += 1

            if d1 <= 0:  # горизонтальный шаг

                d = d + 2 \* x + 1

            else:  # диагональный шаг

                y -= 1

                d = d + 2 \* (x - y + 1)

            continue

        if d > 0:  # пиксель лежит вне окружности

            d2 = 2 \* (d - x) - 1

            y -= 1

            if d2 > 0:  # вертикальный шаг

                d = d - 2 \* y + 1

            else:  # диагональный шаг

                x += 1

                d = d + 2 \* (x - y + 1)

            continue

        if d == 0:  # пиксель лежит на окружности

            x += 1  # диагональный шаг

            y -= 1

            d = d + 2 \* (x - y + 1)

    return 0

def draw\_circle\_mid(self, xc, yc, r):

    x = 0  # начальные значения

    y = r

    p = 5 / 4 - r  # (x + 1)^2 + (y - 1/2)^2 - r^2

    while True:

        draw\_pix(self, xc + x, yc + y)

        draw\_pix(self, xc + x, yc - y)

        draw\_pix(self, xc - x, yc + y)

        draw\_pix(self, xc - x, yc - y)

        draw\_pix(self, xc + y, yc + x)

        draw\_pix(self, xc + y, yc - x)

        draw\_pix(self, xc - y, yc + x)

        draw\_pix(self, xc - y, yc - x)

        x += 1

        if p < 0:  # средняя точка внутри окружности, ближе верхний пиксел, горизонтальный шаг

            p += 2 \* x + 1

        else:  # средняя точка вне окружности, ближе диагональный пиксел, диагональный шаг

            y -= 1

            p += 2 \* (x - y) + 1

        if x > y:

            break

    return 0

def draw\_circle\_std(self, xc, yc, r):

    self.canvas.create\_oval(xc-r, yc-r, xc+r, yc+r, outline=self.color\_pen)

    return 0

# ==========ELLIPSES==========

def draw\_ellipse\_canon(self, xc, yc, a, b):

    for x in range(0, a + 1, 1):

        y = round(b \* sqrt(1 - (x/a)\*\*2))

        draw\_pix(self, xc + x, yc + y)

        draw\_pix(self, xc + x, yc - y)

        draw\_pix(self, xc - x, yc + y)

        draw\_pix(self, xc - x, yc - y)

    return 0

def draw\_ellipse\_param(self, xc, yc, a, b):

    m = max(a, b)

    len = round(pi \* m / 2)

    for i in range(0, len + 1, 1):

        x = round(a \* cos(i / m))

        y = round(b \* sin(i / m))

        draw\_pix(self, xc + x, yc + y)

        draw\_pix(self, xc + x, yc - y)

        draw\_pix(self, xc - x, yc + y)

        draw\_pix(self, xc - x, yc - y)

    return 0

def draw\_ellipse\_br(self, xc, yc, a, b):

    x = 0  # начальные значения

    y = b

    a2 = a \*\* 2

    b2 = b \*\* 2

    d = round(b2 / 2 - 2 \* a2 \* b + a2 / 2)

    while y >= 0:

        draw\_pix(self, xc + x, yc + y)

        draw\_pix(self, xc + x, yc - y)

        draw\_pix(self, xc - x, yc + y)

        draw\_pix(self, xc - x, yc - y)

        if d < 0:  # пиксель лежит внутри эллипса

            d1 = 2 \* (d + a2 \* y) - a2

            x += 1

            if d1 <= 0:  # горизотальный шаг

                d = d + 2 \* b2 \* x + b2

            else:  # диагональный шаг

                y -= 1

                d = d + 2 \* (b2 \* x -  a2 \* y )+ a2 + b2

            continue

        if d > 0:  # пиксель лежит вне эллипса

            d2 = 2 \* (d - b2 \* x) - b2

            y -= 1

            if d2 > 0:  # вертикальный шаг

                d = d - 2 \* a2 \* y + a2

            else:  # диагональный шаг

                x += 1

                d = d + 2 \* (b2 \* x - a2 \* y) + a2 + b2

            continue

        if d == 0:  # пиксель лежит на окружности

            x += 1  # диагональный шаг

            y -= 1

            d = d + 2 \* (b2 \* x - a2 \* y) + a2 + b2

    return 0

def draw\_ellipse\_mid(self, xc, yc, a, b):

    x = 0  # начальные положения

    y = b

    a2 = a \*\* 2

    b2 = b \*\* 2

    p = b2 - a2 \* b + 0.25 \* a2  # начальное значение параметра принятия решения в области tg<1

    while 2 \* b2 \* x < 2 \* a2 \* y:  # пока тангенс угла наклона меньше 1

        draw\_pix(self, xc + x, yc + y)

        draw\_pix(self, xc + x, yc - y)

        draw\_pix(self, xc - x, yc + y)

        draw\_pix(self, xc - x, yc - y)

        x += 1

        if p < 0:  # средняя точка внутри эллипса, ближе верхний пиксел, горизонтальный шаг

            p += 2 \* b2 \* x + b2

        else:  # средняя точка вне эллипса, ближе диагональный пиксел, диагональный шаг

            y -= 1

            p += 2 \* b2 \* x - 2 \* a2 \* y + b2

    p = b2 \* (x + 0.5) \*\* 2 + a2 \* (y - 1) \*\* 2 - a2 \* b2

    # начальное значение параметра принятия решения в области tg>1 в точке (х + 0.5, y - 1) полседнего положения

    while y >= 0:

        draw\_pix(self, xc + x, yc + y)

        draw\_pix(self, xc + x, yc - y)

        draw\_pix(self, xc - x, yc + y)

        draw\_pix(self, xc - x, yc - y)

        y -= 1

        if p > 0:

            p -= 2 \* a2 \* y + a2

        else:

            x += 1

            p += 2 \* b2 \* x - 2 \* a2 \* y + a2

    return 0

def draw\_ellipse\_std(self, xc, yc, a, b):

    self.canvas.create\_oval(xc - a, yc - b, xc + a, yc + b, outline=self.color\_pen)

    return 0

#testing

from math import cos, sin, pi, sqrt

from time import time

# ==========CIRCLES TEST==========

def test\_circle\_canon(xc, yc, r):

    time\_beg = time()

    for x in range(0, r + 1, 1):

        y = round(sqrt(r \*\* 2 - x \*\* 2))

        pix\_x = xc + x; pix\_y = yc + y

        pix\_x = xc + x; pix\_y = yc - y

        pix\_x = xc - x; pix\_y = yc + y

        pix\_x = xc - x; pix\_y = yc - y

    # for y in range(0, r + 1, 1):

    #     x = round(sqrt(r \*\* 2 - y \*\* 2))

    #     pix\_x = xc + x; pix\_y = yc + y

    #     pix\_x = xc + x; pix\_y = yc - y

    #     pix\_x = xc - x; pix\_y = yc + y

    #     pix\_x = xc - x; pix\_y = yc - y

    time\_end = time()

    return time\_end - time\_beg

def test\_circle\_param(xc, yc, r):

    time\_beg = time()

    len = round(pi \* r / 2)

    if len == 0:

        return -1

    for i in range(0, len + 1, 1):

        x = round(r \* cos(i / r))

        y = round(r \* sin(i / r))

        pix\_x = xc + x; pix\_y = yc + y

        pix\_x = xc + x; pix\_y = yc - y

        pix\_x = xc - x; pix\_y = yc + y

        pix\_x = xc - x; pix\_y = yc - y

    time\_end = time()

    return time\_end - time\_beg

def test\_circle\_br(xc, yc, r):

    time\_beg = time()

    x = 0

    y = r

    d = 2 - 2 \* r

    while y >= 0:

        pix\_x = xc + x; pix\_y = yc + y

        pix\_x = xc + x; pix\_y = yc - y

        pix\_x = xc - x; pix\_y = yc + y

        pix\_x = xc - x; pix\_y = yc - y

        if d < 0:

            buf = 2 \* d + 2 \* y - 1

            x += 1

            if buf <= 0:

                d = d + 2 \* x + 1

            else:

                y -= 1

                d = d + 2 \* x - 2 \* y + 2

            continue

        if d > 0:

            buf = 2 \* d - 2 \* x - 1

            y -= 1

            if buf > 0:

                d = d - 2 \* y + 1

            else:

                x += 1

                d = d + 2 \* x - 2 \* y + 2

            continue

        if d == 0.0:

            x += 1

            y -= 1

            d = d + 2 \* x - 2 \* y + 2

    time\_end = time()

    return time\_end - time\_beg

def test\_circle\_mid(xc, yc, r):

    time\_beg = time()

    x = 0

    y = r

    p = 5 / 4 - r

    while True:

        pix\_x = xc + x; pix\_y = yc + y

        pix\_x = xc + x; pix\_y = yc - y

        pix\_x = xc - x; pix\_y = yc + y

        pix\_x = xc - x; pix\_y = yc - y

        x += 1

        if p < 0:

            p += 2 \* x + 1

        else:

            p += 2 \* x - 2 \* y + 5

            y -= 1

        if x > y:

            break

    time\_end = time()

    return time\_end - time\_beg

def test\_circle\_std(xc, yc, r):

    time\_beg = time()

    for i in range (r):

        pix\_x = xc - r; pix\_y = yc - r

        # pix\_x = xc + r; pix\_y = yc + r

    time\_end = time()

    return time\_end - time\_beg

from algorithms\_test import \*

# Draws graphs of time efficiency of different algs

def time\_test(n):

    circles = []

    time\_canon = []

    time\_param = []

    time\_br = []

    time\_mid = []

    time\_std = []

    r\_mas = []

    for r in range(1, n, 10000):

        circles.append([5000, 5000, r])

        r\_mas.append(r)

    for c in circles:

        time\_canon.append(test\_circle\_canon(c[0], c[1], c[2]))

        time\_param.append(test\_circle\_param(c[0], c[1], c[2]))

        time\_br.append(test\_circle\_br(c[0], c[1], c[2]))

        time\_mid.append(test\_circle\_mid(c[0], c[1], c[2]))

        time\_std.append(test\_circle\_std(c[0], c[1], c[2]))

    return time\_canon, time\_param, time\_br, time\_mid, time\_std, r\_mas