МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторная работа №4
По дисциплине “**Надёжность программного обеспечения**”
Тема: “**Оценка надежности с использованием метрик**”

Выполнил:
студент группы ПО-11
Сымоник И.А.
Проверил:
Козик И. Д.

Брест 2025

Цель: Изучить метрики надежности программного обеспечения и применить их для оценки качества кода.

Задание: Поиск медианы в массиве: Реализуйте программу для нахождения медианы в массиве. Проведите тестирование на массивах разного размера.
Изучение метрики надёжности программного обеспечения на основе задания из второй лабораторной работы:

Ход работы

Код программы:

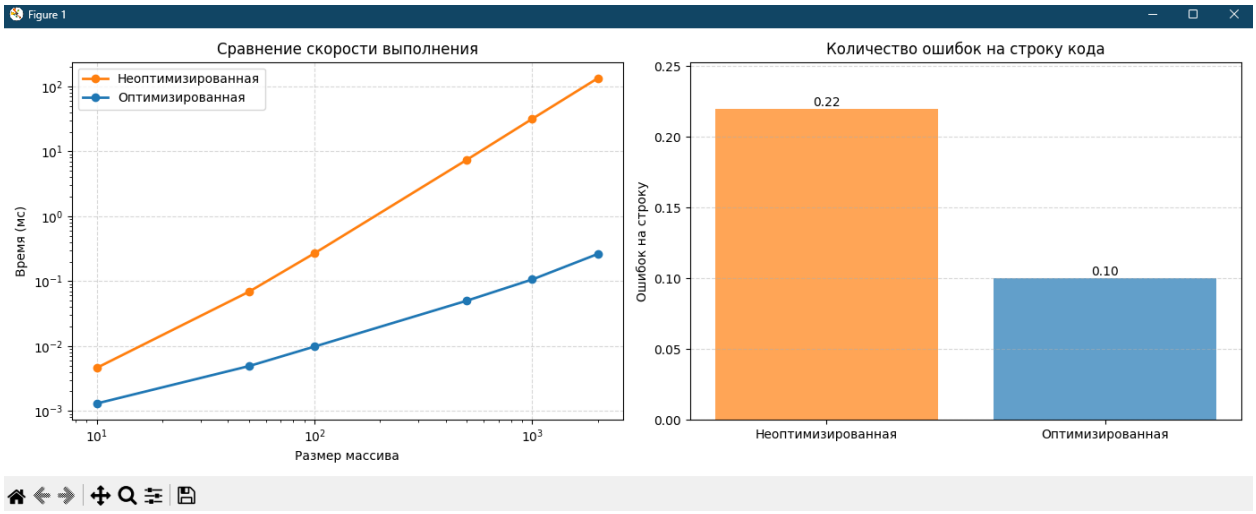
```
import matplotlib.pyplot as plt
import timeit
import random
from typing import List, Union, Callable

def median_unoptimized(arr):
    if not arr:
        return None

    for x in arr:
        if not isinstance(x, (int, float)):
            raise TypeError("Массив должен содержать только числа")

    sorted_arr = arr.copy()
    n = len(sorted_arr)
    for i in range(n):
        for j in range(n-i-1):
            if sorted_arr[j] > sorted_arr[j+1]:
                sorted_arr[j], sorted_arr[j+1] = sorted_arr[j+1], sorted_arr[j]

    mid = len(sorted_arr) // 2
    if len(sorted_arr) % 2 == 1:
        return sorted_arr[mid]
    return (sorted_arr[mid-1] + sorted_arr[mid]) / 2

def median_optimized(numbers: List[Union[int, float]]) -> Union[float, None]:
    if not numbers:
        return None

    if not all(isinstance(x, (int, float)) for x in numbers):
        raise TypeError("Все элементы должны быть числами")

    sorted_nums = sorted(numbers)
    mid = len(sorted_nums) // 2
    return sorted_nums[mid] if len(sorted_nums) % 2 == 1 else (sorted_nums[mid-1] +
sorted_nums[mid]) / 2

def generate_test_data(size: int) -> List[float]:
    return [random.uniform(0, 1000) for _ in range(size)]

def benchmark(func: Callable, data: List[float], repeats: int = 50) -> float:
    timer = timeit.Timer(lambda: func(data))
    times = timer.repeat(repeat=repeats, number=1)
    return min(times) * 1000  # мс

def run_benchmarks():
    sizes = [10, 50, 100, 500, 1000, 2000]
    functions = [
        ("Неоптимизированная", median_unoptimized),
        ("Оптимизированная", median_optimized)
    ]

    results = {name: [] for name, _ in functions}

    for size in sizes:
        data = generate_test_data(size)
```

```
        for name, func in functions:
            time = benchmark(func, data)
            results[name].append(time)

    return sizes, results

ERRORS_METRICS = {
    "Неоптимизированная": {
        "errors_per_line": 0.22,
        "color": "#FF7F0E"
    },
    "Оптимизированная": {
        "errors_per_line": 0.10,
        "color": "#1F77B4"
    }
}

def plot_combined(sizes, perf_results, errors_metrics):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

    for name in perf_results:
        ax1.plot(sizes, perf_results[name], 'o-',
                color=errors_metrics[name]["color"],
                linewidth=2,
                label=name)

    ax1.set_title('Сравнение скорости выполнения')
    ax1.set_xlabel('Размер массива')
    ax1.set_ylabel('Время (мс)')
    ax1.set_xscale('log')
    ax1.set_yscale('log')
    ax1.grid(True, linestyle='--', alpha=0.5)
    ax1.legend()

    names = list(errors_metrics.keys())
    errors = [errors_metrics[name]["errors_per_line"] for name in names]
    colors = [errors_metrics[name]["color"] for name in names]

    bars = ax2.bar(names, errors, color=colors, alpha=0.7)
    for bar in bars:
        height = bar.get_height()
        ax2.text(bar.get_x() + bar.get_width()/2, height,
                f'{height:.2f}',
                ha='center', va='bottom')

    ax2.set_title('Количество ошибок на строку кода')
    ax2.set_ylabel('Ошибок на строку')
    ax2.set_ylim(0, max(errors)*1.15)
    ax2.grid(True, axis='y', linestyle='--', alpha=0.5)

    plt.tight_layout()
    plt.savefig('median_benchmark.png', dpi=120)
    plt.show()

if __name__ == "__main__":
    sizes, perf_results = run_benchmarks()
    plot_combined(sizes, perf_results, ERRORS_METRICS)
```

До улучшения	
Метрика	**Значение**
Цикломатическая сложность	**6**
Ошибок/строку	**0.22**
Временная сложность	**O(n^2)**
После улучшения	

Метрика	Значение
Цикломатическая сложность	**3**
Ошибок/строку	**0.10**
Временная сложность	**O(n * logn)**

```
PS C:\rep\spp\npo4> radon cc main.py
main.py
    F 6:0 median_unoptimized - B
    F 27:0 median_optimized - A
    F 75:0 plot_combined - A
    F 47:0 run_benchmarks - A
    F 39:0 generate_test_data - A
    F 42:0 benchmark - A
```



Вывод: разработали отказоустойчивую систему и провели анализ ее поведения при сбоях.