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)** |

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, снимок экрана, диаграмма, График

Контент, сгенерированный ИИ, может содержать ошибки.**

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